UNIVERSIDAD POLITÉCNICA DE MADRID

FACULTAD DE INFORMÁTICA

TESIS DE MÁSTER

MÁSTER UNIVERSITARIO EN SOFTWARE Y SISTEMAS

IMPLEMENTACIÓN DE UNA PRÁCTICA VIRTUAL DE BIOTECNOLOGÍA: LA TRANSFORMACIÓN BACTERIANA

Autor : Alberto Roque Gamarra

Director : Dr. Jaime Ramírez Rodríguez

MADRID - JULIO DEL 2013

ABSTRACT

This document makes an implementation of the "Bacterial Transformation" phase in the UPM's "Virtual Laboratory of Agroforestry Biotechnology". The current fase represents a continuation of a previous work, in which the virtual laboratory and an architecture for the development of subsequent phases were implemented. The bacterial transformation is the third phase of the genetic modification process of a poplar for giving it resistance against some fungi, and the goal of this phase consists in introducing the genetic modified plasmid in an *Agrobacterium tumefaciens*. For developing this phase, a bunch of actions where determined and added to the automatic tutor. In adition, some scripts were developed for some 3D objects already existing and new 3D instruments and machines were modelled for the new phase. On other hand, after the testing phase of the two first phases, some issues related to usability had to be fixed in the Firestorm viewer. This viewer is an open-source project then the code is available in the author website. Download and modification of this source code were made for fixing the identified problems. The solution of this issues are part of this work.

RESUMEN

En este documento se realiza la implementación de la fase "Transformación Bacteriana" en el "Laboratorio Virtual de Biotecnología Agroforestal" de la Universidad Politécnica de Madrid. Esta fase representa una continuación de un trabajo previo, en el que se implementó el laboratorio virtual y se diseñó una arquitectura para el desarrollo de las fases posteriores. La Transformación Bacteriana es la tercera fase del proceso de modificación genética de un chopo para dotarle de resistencia frente a ciertos hongos y tiene como objetivo la introducción de un plásmido modificado genéticamente en la bacteria *Agrobacterium tumefaciens*. Para el desarrollo de esta fase se determinaron las acciones que debían de ser agregadas al tutor automático y, además, se programaron los *scripts* de ciertos objetos 3D ya existentes y se modelaron en 3D nuevos instrumentos y maquinarias necesarios para la realización de la práctica. Por otra parte, luego de la etapa de pruebas de las dos primeras fases, se concluyó que se debían de resolver algunos problemas de usabilidad del visor Firestorm. Gracias a que este visor es un proyecto de código abierto, fue posible corregir los problemas identificados. La resolución de estos problemas se explica como parte del presente trabajo.

Índice general

Índice de cuadros

Índice de figuras

AGRADECIMIENTOS

Quisiera agradecer a:

mi familia por haberme apoyado en cada momento de mi vida y por haberme dado ánimos cuando más lo necesitaba durante mi estancia en Madrid.

mis amigos por apoyarme en la realización de este máster y por brindarme su amistad durante todos estos años.

Jaime Ramírez por la confianza hacia mi persona y por haberme orientado y apoyado en la realización de este trabajo.

Angélica de Antonio por la confianza depositada en mi persona y por todo el conocimiento transmitido.

Miembros del Laboratorio Decoroso Crespo por haberme acogido tan amablemente y en especial a Diego y a Daniel por ayudarme con el aprendizaje de OpenSim.

la Universidad Politécnica de Madrid por haberme permitido estar en sus aulas durante todos estos meses y por toda la enseñanza brindada.

DEDICATORIA

A mi padre, a mi madre, a mi hermana y a mis abuelitos que me cuidan desde el cielo y siempre han deseado lo mejor para mí.

Capítulo 1

Introducción

Durante las últimas décadas, las herramientas de software han brindado diversas soluciones en distintos campos, con el objetivo de minimizar costos y tiempo. Dado que se cuenta con recursos limitados, es preciso que se pueda sacar el mayor provecho de ellos. Mediante el desarrollo de software es posible cumplir con ello; sin embargo, si bien es cierto que los sistemas operacionales cubren las mayorías de las necesidades en las empresas o instituciones, algunas veces se requiere un mayor interés por parte de los usuarios, por tanto es importante motivarlos para que de alguna manera puedan sentirse más ligados al software.

Dentro del campo educativo se hacen innovaciones continuas para que el alumno pueda aprender de una manera más rápida, mejorando las técnicas de enseñanza; si hace algunos años se estudiaba con libros y cuadernos, actualmente, el uso masivo del computador ha tenido un impacto muy significativo y día a día cambian la pedagogía. En aulas de clase tradicionales, el profesor interactúa con los alumnos de manera directa, promoviendo su participación, y a pesar de ello, algunos estudiantes no se sienten cómodos expresando sus opiniones y eso provoca que no se abra una discusión más abierta entre ellos.

Por otro lado, el mundo de los videojuegos y las consolas tienen mucha acogida por parte de los jóvenes, y aún más, cuando sus características los hacen más realistas. Las personas se ven inmersas en un mundo fantasioso y sin limitaciones. Los tipos de videojuegos en los que es posible controlar un personaje (avatar) ya llevan muchos años entre nosotros y podría decirse que una de sus grandes áreas de influencia ha sido la de los mundos virtuales. Un mundo virtual no es más que un entorno similar a la realidad, un universo en el que todo puede ser posible, en el que el ritmo del tiempo puede pasar tan lentamente o tan rápidamente como deseemos; además, puede estar lleno personajes irreales y pueden pasar cosas que en nuestro mundo real sería imposible. En los últimos años, los mundos virtuales han facilitado también la interacción social; se podría mencionar que

Second Life es una de las plataformas más conocidas en este ámbito.

Sin embargo, la socialización y los negocios no son los únicos objetivos que se pueden perseguir en este tipo de mundos. En el presente documento se trata de mejorar la enseñanza para los alumnos de un curso de Biotecnología a través de un laboratorio virtual. Teniendo como punto de partida un trabajo anterior [Riofrío Luzcano, 2012] en el que se implementó un laboratorio de Biotecnología, el material y el instrumental necesario para que los estudiantes puedan realizar prácticas con numerosas ventajas. Por ejemplo, una de las ventajas más representativas y que justifica su construcción, es el ahorro en los costes de realización de las prácticas ya que en un semestre, un alumno podría gastar un aproximado de entre 10 000€ y 24 000€ en materiales de laboratorio reales; además, otra ventaja del laboratorio virtual frente al real es que permite evitar las esperas asociadas a la ejecución de la práctica como, por ejemplo, las esperas requeridas para conseguir el crecimiento de algunas plantas o microorganismos. Por otra parte, los estudiantes no están expuestos al peligro de ciertos componentes químicos y no existe el riesgo de dañar la maquinaria del laboratorio. Finalmente, los alumnos siempre podrían realizar la práctica real una vez que hayan comprendido el proceso completo y entiendan lo que se debe de hacer en cada momento.

Para el desarrollo de entornos educativos en los mundos virtuales, se dispone de plataformas como SecondLife u OpenSim, si bien la segunda es la preferida debido a que es de código abierto y tiene la posibilidad de instalarse en un servidor de la propia institución académica. OpenSim tiene las mismas posibilidades que SecondLife, pero no requiere ningún pago por el servicio, tan sólo el coste que incluye el hardware y su mantenimiento. En ambos casos, para poder acceder al mundo virtual se necesita de un visor o cliente que se encargue del renderizado de los objetos 3D y de su descarga desde el servidor cuando el avatar comienza a navegar por el mundo. Para el trabajo actual se hizo uso del visor Firestorm por su estupendo rendimiento y porque era de código abierto, lo que hacía factible que se le puedan hacer modificaciones para mejorar su usabilidad.

Este documento representa una continuación del trabajo desarrollado en la tesis de máster de Diego Riofrío [Riofrío Luzcano, 2012], que surgió de la necesidad de que alumnos del curso de Biotecnología puedan realizar una práctica en la que tienen que transformar genéticamente un chopo para dotarle de resistencia frente a ciertos hongos. Para cumplir este objetivo, se construyó un laboratorio de Biotecnología con todos sus instrumentos y aparatos necesarios utilizando la plataforma OpenSim. Además, para validar las acciones que se realizan durante la práctica, se diseñó un tutor automático que funciona con una configuración preestablecida para cada fase, este tutor brinda mensajes a los alumnos durante toda la práctica, a modo de ayuda y como orientación para las siguientes acciones.

Además de este tutor se diseñaron la primera y la segunda fase de la práctica, denominadas "Micropropagación de Material Vegetal" y "PCR (Reacción en Cadena de Polimerasa)".

Como un trabajo posterior y basado en [Riofrío Luzcano, 2012], en [Pedraza, 2013] se implementó una fase más de la práctica, la fase de la "Ligación". En este nuevo trabajó se hizo una reutilización de los instrumentos o aparatos ya construidos y en la configuración del tutor se agregaron las acciones pertinentes para el desarrollo de esta nueva fase.

En este trabajo se realizará el diseño y programación de la fase "Transformación Bacteriana", fase posterior a la "Ligación"[Pedraza, 2013], en el que se introducirá un plásmido modificado genéticamente (en la fase anterior) en la bacteria *Agrobacterium tumefaciens*. Para este fin se han construido diversos objetos que no se encontraban en el laboratorio, como el aparato de electroporación, la cubeta de electroporación y una fuente de voltaje. Además, se describirá el proceso que deben de seguir los alumnos para completar esta práctica. De igual manera que en [Riofrío Luzcano, 2012] y en [Pedraza, 2013], se han agregado las acciones necesarias en la configuración del tutor, además de las dependencias y los mensajes asociadas a estas acciones que se mostrarán a los usuarios.

En el capítulo "Estado de la Cuestión" se hará una descripción de los mundos virtuales, así como las plataformas de desarrollo más utilizadas en el ámbito educativo, Second Life y OpenSim. Además, se mencionarán los lenguajes de desarrollo en ambas plataformas y se hará una comparación entre ellas para determinar cuál es la que mejor se adapta a un fin educativo. Finalmente, se mencionarán algunas aplicaciones educativas que se han realizado en otros trabajos similares.

En el capítulo "Planteamiento del Problema" se describirá someramente un laboratorio de biotecnología real y algunas de las técnicas que son comúnmente utilizadas en la Biotecnología. Además, se mencionará el proceso completo de la práctica, detallando todas sus fases y el objetivo de cada una de ellas, pero prestando una especial atención a la Transformación Bacteriana. Por otra parte, como se mencionó en la sección de "Trabajos Futuros" de [Riofrío Luzcano, 2012], las pruebas con los profesores e investigadores de Biotecnología estaban pendientes. Sin embargo, cuando se inició el presente trabajo ya se habían realizado dichas pruebas y una de las concluiones a las que se llegó fue que se debía mejorar la usabilidad del visor Firestorm. En este capítulo también se describen todos los problemas de usabilidad identificados en el visor y qué se debería cambiar en la interfaz gráfica del visor para que sean resueltos.

En el capítulo "Solución Adoptada", se explicará cómo se implementó la "Transformación Bacteriana", es decir, se explicará la configuración del tutor así como el diseño de los objetos activos que participan en esta fase de la práctica. Además, se dará solución a los problemas de usabilidad de Firestorm, abordando cada problema que se describió en

el capítulo “Planteamiento del Problema”.

En el capítulo final se explicarán las conclusiones a las que se ha llegado tras el desarrollo de la “Transformación Bacteriana” y de las modificaciones en el visor Firestorm. Finalmente, se plantearán algunas ideas como trabajo futuro que se podrían derivar del presente trabajo.

Capítulo 2

Estado de la Cuestión

En este capítulo se introducirán los mundos virtuales, sus características y las ventajas que tienen; se mencionarán algunas de las plataformas de desarrollo de mundos virtuales más relevantes que han sido utilizadas para crear entornos de aprendizaje, prestando especial atención a OpenSim y finalmente, se mostrarán algunas aplicaciones educativas creadas con OpenSim.

2.1. Introducción a los Mundos Virtuales

Hoy en día, un mundo virtual podría ser definido [Castronova, 2011] como un entorno similar a un mundo real, con capacidad de expandirse y, en algunas ocasiones, sin algunas leyes físicas; además, es creado por humanos y es mantenido por un computador.

En los mundos virtuales, los usuarios suelen ser representados a través de personajes humanoides llamados avatares. Cada avatar tiene la capacidad de caminar, correr, volar, teletransportarse, etc; además, puede interactuar con otros, a través de una ventana de chat, etc.

Dentro de un mundo virtual, las personas reales, a través de sus *avatares*, pueden interactuar y establecer relaciones sociales dentro de él. Pero como en un mundo real, no sólo es posible permitir las relaciones sociales, sino que se pueden diseñar y construir estructuras como escuelas, compañías, oficinas, hospitales, centros recreativos, entre otros. Si bien es cierto que los mundos virtuales no surgieron con todas esas posibilidades, con el tiempo, a través del avance tecnológico, su representación de la realidad ha sido mucho más fiel.

En su libro *"Designing Virtual Worlds"*, Barton [Bartle, 2003] clasifica la evolución de los mundos virtuales en cinco etapas, la etapa más temprana puede remontarse a 1978, en la que el juego de rol *on-line* MUD(*Multi-User Dungeon*) fue el primero que se cons-

truyó bajo una arquitectura similar a los mundos virtuales actuales. MUD fue desarrollado por Roy Trubshaw (1978), en lenguaje ensamblador MACRO-10 en la Universidad de Essex (Inglaterra). Esta versión de MUD era un mundo virtual basado en texto; posteriormente, fue agregado el soporte para gráficos con lo que su nombre cambió a "*Massively-Multiplayer Online Role-Playing Games*" (MMORPG)[Riofrío Luzcano, 2012].

En las siguientas etapas de la evolución, se cambió la perspectiva de ser utilizados sólo como juegos de rol, y así surgieron mundos en los que primaba la fantasía y se vió la posibilidad de ganar dinero a través de ellos; en 1989, por ejemplo, "TinyMUD" y "Zone" fueron lanzados y marcaron un hito importante ya que fueron los primeros mundos virtuales sociales; posteriormente le sucedieron *Second Life*, *Haboo Hotel* y *There*.

Con el avance de los mundos virtuales, y con la factibilidad de poder crear realidades que eran limitadas sólo por la imaginación del creador, éstas se vieron influenciadas por obras literarias como *"The Lord of The Rings"*, *"The Wheel of Time"*, *"Discworld"*, *"Dungeons & Dragons"*, entre otros; además de series de televisión o películas tales como *"Star Wars"* o *"Battlestar Galactica"*.

Posteriormente, la influencia de los juegos de rol llevó el mundo del entretenimiento a los mundos virtuales y en los que, en los últimos años, las consolas han sabido ganar terreno. En la actualidad, existen diversidad de mundos virtuales, cada uno de ellos con los enfoques que se mencionaron anteriormente. Dentro de ellos, los más populares son *World Of Warcraft*[Inc., 2013], *Second Life*[Inc, 2013b], *The Sims*[Inc, 2013a],*Rune Scape*[Ltd, 2013], *Gaia*[Studio, 2013], entre otros.

La arquitectura [Allison et al., 2010] de un mundo virtual multiusuario es cliente-servidor y funciona de manera similar a una aplicación web. Se necesita un visor (cliente) que, generalmente, está disponible en la página del proveedor y un servidor que es el que administra todo el mundo virtual. Dado el alto renderizado de los objetos del mundo, el ordenador en el que se ejecuta el cliente debe de contar con una buena memoria gráfica para poder proporcionar una sensación satisfactoria al usuario.

2.2. Creación de Mundos Virtuales

2.2.1. Plataformas de desarrollo

Como se mencionó en 2.1, existen diversidad de mundos virtuales con distintos enfoques. Para el desarrollo de entornos educativos hay dos que destacan, *Second Life* y *Open Sim*; aunque el primero fue diseñado para ser un entorno social abierto a todo el mundo, ha sido adoptado por instituciones educativas como universidades y escuelas.

Second Life

Second Life (SL)[Rymaszewski, 2007] fue desarrollado por Linden Lab, cuya fundación se remonta a 1999. Inicialmente, esta compañía se dedicaba a la investigación y desarrollo de *haptics*[1], pero posteriormente esta línea de negocio fue abandonada para concentrarse en los mundos virtuales.

En una primera etapa, Second Life fue llamado LindenWorld y empezó a ser desarrollado en el 2001 debido a las necesidades de testing de los *haptics*. En esta etapa no fue abierta al público, sino que se enfocó como un juego de tipo *shooter-game* en que los avatares fueron construidos mediante la utilización de prims[2].

Posteriormente, LindenWorld culminó su fase alfa y fue llamado Second Life. En esta etapa, ya se contaba con un mundo virtual 3D de mayor robustez y en la que los usuarios podían interactuar con otros en tiempo real. En octubre del 2002 se lanza una versión beta con 16 regiones en la que Steller Sunshine fue el primer residente[3] en unirse. Al año siguiente, se produce el lanzamiento oficial en la que ya aparecían, por primera vez, los Linden Dollar (L$).

Dentro de Second Life, los residentes suelen llamar a las regiones *Sims* (de simuladores) debido a que, originalmente, un servidor alojaba sólo una región. Este servidor mantiene un seguimiento de varios objetos del mundo, tales como los avatares, las parcelas, las regiones, el estado del inventario, entre otros.

Dentro del ámbito educativo, las ventajas de Second Life son tales que, permiten lo que un espacio real no lo conseguiría, sobre todo, la eliminación de barreras entre el profesor y el alumno. En este tipo de entornos, un estudiante puede sentirse más cómodo y puede expresar sus opiniones y fomentar discusiones con mayor facilidad, apoyándose en una educación a distancia. Además, la disponibilidad de recursos de aprendizaje o herramientas que el entorno puede ofrecer, fortalecen las estrategias pedagógicas dentro de las instituciones educativas [Savin-Baden, 2010].

[1]Dispositivos táctiles con las que una persona se puede poner en contacto con un entorno virtual a través de movimientos, vibraciones o fuerzas hechas por sí mismo.

[2]También llamado primitiva, es un objeto único que sirve como base para la construcción de objetos más complejos.

[3]Usuario en Second Life, también llamado *Resi*.

Figura 2.1: Ayuda en la Comunidad de la Universidad de Oxford en Second Life.

Second Life ha sido adoptado para la creación de entornos educativos como aulas virtuales o laboratorios de física, química, etc., además, algunas universidades lo usan como *"front-of-house"*[4] o como entornos de ayuda (figura 2.1)[5] en los que se brinda orientación a los alumnos o se utiliza como centro de informes de las actividades o eventos de la universidad. En [Allison et al., 2010] destacan ciertas observaciones que deben ser consideradas para la creación de entornos de aprendizaje:

- El lenguaje nativo de desarrollo es de tipo *script* y se llama *Linden Scripting Language* (LSL). Dado que tiene muchas limitaciones, sería preferible el uso de lenguajes mucho más potentes como Java o C#, que ofrecen un *framework* más sólido.
- Es gratuito para crear avatares y navegar en el mundo. Sin embargo, es de uso comercial para los que deseen adquirir su propio terreno, lo que supone una inversión para crear entornos de aprendizaje.
- Debido a su arquitectura, la distribución de los servidores en varios países puede aumentar la latencia y disminuir los tiempos de renderizado del mundo; asimismo, esto también depende del ancho de banda del cliente.
- Una de las condiciones para registrarse en SL, es ser mayor de 18 años, pues existe contenido para adultos al que se puede acceder libremente. Dado que los estudiantes

[4]Concepto similar a una página web oficial.
[5]https://d1yjxggot69855.cloudfront.net/images/2/23/Basic_mode.png

de las escuelas no superan esta edad, puede resultar un inconveniente; además, la distracción con ese tipo de contenidos debe de mantenerse alejada.

OpenSimulator

OpenSimulator (también llamado OpenSim) es un servidor de aplicaciones de código abierto, multi-plataforma y multi-usuario que soporta el desarrollo y el alojamiento de mundos virtuales [OpenSim, 2013b]. Según lo mencionado en la Wiki de OpenSim [OpenSim, 2013b], el proyecto fue fundado por Darren Guard en el 2007 [OpenSim, 2013a] gracias a que en Enero de ese mismo año, el cliente de Second Life fue liberado como código abierto con el objetivo de crear clientes gráficos personalizados que se pudieran conectar a su servidor.

El lenguaje de desarrollo por defecto, al igual que Second Life, es LSL, pero además, es posible la utilización de C#, J# o VBScript (los dos últimos funcionan sólo en Microsoft Windows), si bien estos últimos presentan algunos problemas de seguridad. OpenSim ha sido escrito en C#, por lo que para ejecutarse en Microsoft Windows se requiere tener instalado el .NET Framework 3.5 o superior y para Linux o MacOS, Mono Framework 2.4.3 o superior. Su código fuente está disponible bajo la licencia BSD (Berkeley Software Distribution)[OpenSim, 2013b].

OpenSim necesita de un gestor de base de datos relacional para poder almacenar cierta información del mundo. Por defecto soporta SQLite y puede ser usada sin configuraciones adicionales. Además, tiene soporte completo para MySQL desde la versión 5.1, y es la opción recomendada para entornos de producción. Finalmente, también tiene soporte para Microsoft SQL Server, aunque algunas funcionalidades aún no se encuentran terminadas.

La arquitectura de OpenSim está basada en *plug-ins* en el micro-kernel[Sun et al., 2010]. El micro-kernel es la plataforma de ejecución, se encarga de crear, inicializar, actualizar y cerrar cada región mediante la carga de un *plug-in*. La configuración por defecto carga 62 plug-ins, algunos de ellos se cargan cuando se inicia OpenSim, como por ejemplo, aquél que se encarga de la carga de otros plug-ins o el que administra las regiones del mundo virtual.

En cuanto a la instalación, OpenSim se puede configurar para funcionar en modo Standalone o Grid[OpenSimulator, 2013a]. Como se muestra en la figura 2.2, en el modo Standalone se ejecuta el simulador de la region y todos los servicios de datos en un único proceso llamado *OpenSim.exe* y es posible crear cuantas regiones sean necesarias.

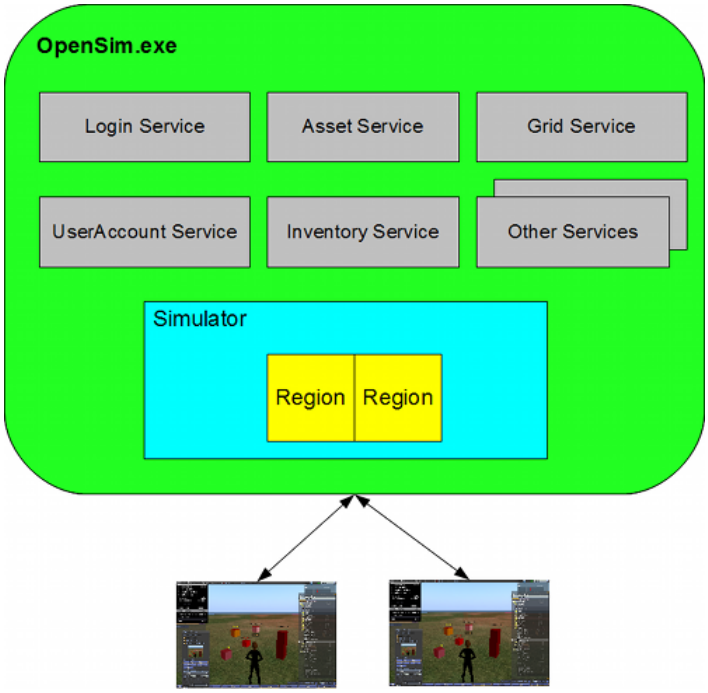

Figura 2.2: OpenSim ejecutándose en modo Standalone.[OpenSimulator, 2013a]

Mientras que en la figura 2.3, en la configuración Grid, los servicios de datos no son parte del proceso del servidor de la región y los simuladores de las regiones pueden ser instalados en varios equipos, de manera que se pueden tener distintas regiones ejecutándose bajo distintos procesos *OpenSim.exe* y mejorar el rendimiento[OpenSimulator, 2013a].

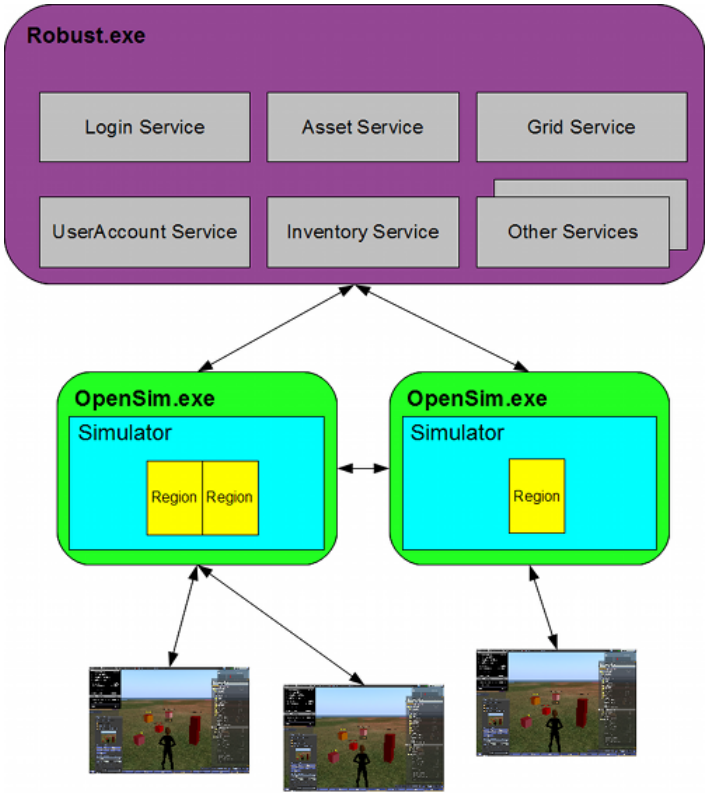

Figura 2.3: OpenSim ejecutándose en modo Grid.[OpenSimulator, 2013a]

De la misma manera que en Second Life, es posible acceder al mundo virtual de OpenSim a través de un visor, a través de él se inicia sesión y se inicia la descarga del último estado del avatar en el mundo; de igual manera, se encarga del renderizado 3D y de la descarga de los objetos en el instante en el que el usuario navega lo suficientemente cerca de ellos. Existen diversos visores para OpenSim y todos ellos soportan las mismas funcionalidades que el visor de Second Life, mensajes de chat o voz, administración de inventarios, construcción de objetos, IM, IM en grupos, entre otros. Entre estos visores se encuentran los siguientes:

- **Firestorm**[Inc., 2013]. Es el sucesor del Phoenix Viewer. Está basado en el código base Linden Lab V3 LGPL. Está desarrollado para sistemas operativos Microsoft Windows, Linux y Mac OS X.

- **Kokua**[Imprudence, 2013]. Es un proyecto abierto basado en el visor abierto de Second Life. Su objetivo es mejorar la usabilidad a través del diseño, la participación de las comunidades y adaptándolo a desarrollos de software modernos.

- **Singularity**[Singularity, 2013]. Es un cliente que también es compatible con Second Life y está basado en el código fuente del visor Ascent, desarrollado por Balesarph Software Group, que, a su vez, fue basado en el visor abierto de Second Life.
- **Radegast Metaverse**[Radegast, 2013]. Es un cliente que está escrito usando *libopenmetaverse*[6] y puede ser ejecutado en cualquier plataforma a través del .NET o Mono Framework.

2.2.2. Lenguajes de desarrollo

Para el desarrollo de mundos virtuales en Second Life u OpenSim se requieren de lenguajes de tipo *script*, son soportados tanto en Microsoft Windows como en Linux y en Mac OS X se mencionan a continuación[OpenSimulator, 2013b].

LSL (Linden Scripting Language)

El lenguaje por defecto de Second Life y OpenSim, se encuentra en su versión 2 y es conocido también como LSLV2 y su sintaxis es similar a C. Como se menciona en [Life, 2013], LSL brinda comportamientos a las primitivas, objetos y avatares. Dentro del lenguaje existen constantes, eventos, sentencias de control (for, if, while, etc), funciones, operadores, tipos de datos, estados, variables y errores.

El estado por defecto de un objeto es el *default* y, dentro de él, su método inicializador es el *state_entry()*. Este estado se inicia cada que se crea o que se reinicia el *script*. Posteriormente, el estado del objeto representado en el *script* puede cambiar, cuando lo decida el propio *script*.

Existen otros métodos para interactuar con un objeto; por ejemplo, el *touch_start* es lanzado cuando el usuario toca el objeto; el *listen*, cuando algún objeto externo le envía un mensaje, o el *timer*, que se ejecuta con una secuencia prefijada y se inicia cuando se ejecuta la función *llSetTimerEvent*.

El LSL dispone de una serie de funciones predefinidas que permiten realizar operaciones de uso habitual. Las funciones LSL mantienen el prefijo *"ll"* como por ejemplo *llGetKey()*, que obtiene el UUID del objeto en el que se está ejecutando el *script*; *llSay()*, *llRegionSay()* o *llWhisper()* para enviar un mensaje a través de un canal de comunicación, pero que se diferencian en el alcance en distancia del mensaje; *llSetPos()* para colocar un objeto en una posición dada (x,y,z).

[6]Colección de librerías .NET usadas para la interacción con simuladores de mundos virtuales en 3D.

Las siguientes líneas corresponden a la sintaxis LSL:

```
default
{
   state_entry()
   {
      llSay(0, "Inicializar objeto.");
   }
}
```

OSSL (OpenSimulator Scripting Language)

Son un complemento a las funciones de Second Life, pero que solamente funcionan en OpenSim. Se usan de la misma manera que las funciones LSL y en muchas ocasiones, permiten escribir menos líneas de código. Estas funciones empiezan por el prefijo *"os"*, como por ejemplo *osTeleportAgent()*, para teletransportar un avatar a otra region; *osSetDynamicTextureURL()* para que la textura de un objeto pueda ser descargada de internet, o *osParseJSON()* para convertir un objeto a formato JSON. Para hacer uso de estas nuevas funciones se debe modificar el fichero de configuración *OpenSim.ini* y reiniciar el servidor.

C# (C Sharp)

Es un lenguaje de desarrollo para la plataforma .NET. Tan solo está disponible en OpenSim. Para que el compilador lo interprete como C#, el fichero del *script* debe empezar por *//C#*. Está soportado en sistemas operativos Microsoft Windows y Linux a través de .NET y Mono Framework, respectivamente. Dada la robustez del lenguaje, se pueden aprovechar muchas librerías que LSL no posee tal como los tipos de datos, listas genéricas, excepciones. Sin embargo existen algunos *namespaces* que tienen problemas de seguridad debido a que pueden "escribir" en el servidor, como por ejemplo, la clase *System.IO.File*[Allison et al., 2010]. Las siguientes líneas corresponden a la sintaxis C#:

```
//c#
public void default_event_state_entry()
{
    llSay(0, "Inicializar objeto.");
}
```

VB .NET (Visual Basic .NET)

De la misma manera que C#, es un lenguaje de la familia .NET pero sólo funciona en Microsoft Windows debido a que Mono Framework no tiene un compilador para VB .NET.

Igualmente, el fichero del *script* debe empezar con *//vb* para que pueda ser compilado. Las siguientes líneas corresponden a la sintaxis VB .NET:

```
//vb
Public Sub default_event_state_entry()
    llSay(0, "Inicializar objeto.")
End Sub
```

2.2.3. Comparación entre Second Life y OpenSim

En esta sección se hará un resumen comparativo entre ambas plataformas; en el cuadro 2.1 se muestran los aspectos más destacables de una comparación que se hace en el paper *"Educationally Enhanced Virtual Worlds"*[Allison et al., 2010], entre una aplicación desarrollada en Second Life y otra, en OpenSim; en ambas se requería que el alumno realice algunos ejercicios de programación en lenguaje LSL, por lo que tenían que codificar, crear objetos 3D y luego, reportarlos al profesor para que puedan ser calificados; al final del documento y de las experiencias en ambos entornos se extrajeron algunas conclusiones que se mencionan a continuación:

Tópico	**Second Life**	**OpenSim**
Costo	Pago anual por región.	Gratuito por ser open source. Se requiere el gasto de hardware.
Tamaño del terreno	No escalable. Basado en el plan de pago.	Limitado por el hardware del servidor.
Edad de usuarios	Mayores de 18 años.	Ninguna.
Calidad del servicio	Puede ser lenta debido a la latencia de la red y de la conexión a Internet.	Reducción de la latencia si la conexión es local.
Lenguaje	Sólo LSL	LSL, C#, VB.

Cuadro 2.1: Comparación entre Second Life y OpenSim.

Del cuadro 2.1 se puede concluir que para el desarrollo de entornos de aprendizaje, es más conveniente el uso de OpenSim, dado que no tiene algunas de las limitaciones que sí tiene SL, puede ser más rápido y gracias a que es de código abierto, su código está disponible y puede ser modificado acorde a las necesidades de las instituciones educativas.

2.3. Aplicaciones Educativas de OpenSim

Dadas las ventajas de OpenSim mencionadas en el cuadro 2.1, en esta sección se describirán varios proyectos en los que ha sido utilizado, en el ámbito del aprendizaje y la enseñanza.

En [Andreas et al., 2010] utilizan OpenSim y Moodle a través del módulo Sloodle con el objetivo de mejorar las técnicas de aprendizaje colaborativo Jigsaw y Fishbowl. El CSCL (Computer-supported collaborative learning) es una de las técnicas más prometedoras en cuanto al aprendizaje y la enseñaza moderna a través de las Tecnologías de la Información y la Comunicación. Adicionalmente, los LMS (Learning Management Systems) ofrecen muchas posibilidades a través de repositorios, las discusiones en los foros, las salas de chat y los reportes de notas a través de los cuestionarios. En este documento se destaca el potencial y la efectividad de VLE (Virtual Learning Environments) en mundos virtuales en 3D, considerando que son un apoyo a la formación del conocimiento, inmersión, interactividad y educación. Mencionan la aplicabilidad de Moodle en entornos virtuales como Second Life y OpenSim; en este último destacan la ventaja de ser de código abierto y de que, para propósitos educativos, puede ser utilizado en un servidor privado; además, no es tan restrictivo como Second Life en el sentido de los permisos, el tamaño del terreno y los scripts. En su investigación examinaron técnicas CL (Collaborative Learning) a través de Sloodle y OpenSim con el objetivo de demostrar la efectividad y aplicabilidad de los LMS y los VLE en 3D.

Aplicaron la técnica de Jigsaw en la que se brinda cierta parte de un conocimiento global a cada alumno que luego es compartido entre ellos. Además, aplicaron la técnica Fishbowl en la que los alumnos se disponen en círculos concéntricos en el que el círculo interior discute cierto tema, y el exterior, observa y escucha. La implementación de esta investigación se encontraba pendiente en la fecha del documento, pero la consideraban como trabajo futuro; sin embargo, destacaban que los VLEs sí ofrecen un valor agregado y esperaban que ambas técnicas se complementaran.

Otro ejemplo de la ventaja que tiene OpenSim por ser de código abierto se encuentra en [María, 2013]. En este artículo, la Armada de los Estados Unidos consideraba que el uso de Second Life era inviable debido a sus necesidades militares. Concretamente necesitaban una plataforma segura que pudiera ser completamente ejecutada en sus servidores, con un acceso restringido y sólo para personal autorizado. Es así como el STTC (Simulation and Training Technological Center) optó por descartar SL y emplear OpenSim. El trabajo recibe el nombre de programa MOSES y empezó el 2010 y continúa hasta la actualidad, comprende un área de 3000 acres cuadrados en todo el mundo virtual al que los usuarios

se conectan con unas credenciales mediante el visor Firestorm.

El programa MOSES (Military Open Simulator Enterprise Strategy) está diseñado para entrenamientos de soldados, reuniones virtuales y para expandir el uso de los mundos virtuales; además, los usuarios pueden sentir una sensación más real y se dan cuenta de que tiene muchas más capacidades que en un videojuego. La verosimilidad llega al punto de que los terrenos son modificados, es decir, si caen bombas, se crean cráteres dependiendo de la intensidad de la explosión. Por otra parte, es posible crear montañas, lagos, ríos y los avatares pueden ser afectados por los objetos del mundo. Finalmente, MOSES puede ser solicitado por aquellas compañías que estén interesadas en probar OpenSim; incluye el servidor, un visor personalizado y un área de 64 acres.

Otra experiencia destacable se menciona en [Lorenzo et al., 2012], en la que se aprovecha el boom de la presencia del español en países como Brasil, China o Estados Unidos, para concebir una plataforma tecnológica que permita lla creación, desarrollo y despliegue de contenidos para la enseñanza del español en un videojuego educativo. El artículo ilustra el proyecto SLRoute, fundado por el programa "Avanza Contenidos" y auspiciado por el Ministerio de Industria.

El componente más importante de SLRoute es el mundo virtual alojado en un servidor OpenSim. En este mundo se construyeron instrumentos para el desarrollo del entrenamiento y materiales de aprendizaje para la descripción de los retos y los objetivos que necesitaban ser cumplidos. Además, contiene herramientas colaborativas como chat de texto y voz, foros, correos electrónicos, etc.

Se construyó una metodología de enseñanza del español, diseñada para cinco ambientes lingüísticos distintos y cada uno con tres niveles de dificultad. Además, se creó un curso de español para visitantes en formato SCORM[7].

OpenSim también ha llegado a las especialidades médicas. En [Datli et al., 2009] se adaptó un entorno de aprendizaje representado en un mundo virtual OpenSim ya existente, con el objetivo de configurarlo y adaptarlo a otras especialidades. Inicialmente, el entorno 3D fue creado para uso cardiológico, y posteriormente fue modificado para las necesidades de la psiquiatría educacional.

La mayoría de los objetos del centro médico ya habían sido desarrollados y sólo quedaba por construir el edificio completo y las salas de consulta. Los avataras fueron también adecuados para cambiar su presentación a los grupos de psicoterapia.

El contenido educacional fue descrito a través de un esquema de metadatos y el objetivo era proveer una descripción precisa de los objetivos y retos del escenario. Para la

[7] *Sharable Content Object Reference Model*. Conjunto de estándares y especificaciones que permite crear objetos pedagógicos.

descripción de estas metas se usó el IEEE LOM Editor, que utiliza el estándar abierto "IEEE 1484.12.1 - 2002 LOM Standard"para la descripción de objetos de aprendizaje. Se describían atributos de los objetos como el autor, el tipo, el propietario, entre otros.

Otra aplicación que cabe destacar se describe en [Berns et al., 2013]. En este paper se desarrolló una aplicación similar a un juego en un entorno virtual y bajo OpenSim en la Universidad Autónoma de Madrid. Se construyó una plataforma llamada **VirtUAM** para permitir el diseño de sistemas de juegos. Está conformado por distintos módulos como el grid que se encarga de la ejecución y administración de los mundos virtuales; un portal web que provee acceso a distintos recursos para estudiantes y profesores; y, por último, la base de datos que almacena información del comportamiento del usuario y la interacción con el mundo virtual.

Durante el juego, los estudiantes se desplazan a través de diversas habitaciones o niveles, en los que encuentran distintos tipos de retos colocados en las paredes. El juego está compuesto por 5 niveles, en el que el nivel 0 sólo consiste en una sala de reunión que brinda ayuda al juego; a partir del nivel 1 empiezan los juegos como el "Training Room", el "Quiz-room", el "Practica room or amusement arcade" y, finalmente, el "Supermarket".

Como consecuencia de la aplicación de estos juegos, se concluyó que hacían que el aprendizaje sea más sencillo y mucho más rápido; además, lo hace más entretenido y divertido. Las características del juego como la competencia y la colaboración motivan al estudiante a superarse a sí mismo.

Cabe destacar que ninguna de las aplicaciones mencionadas funciona con un tutor automático que oriente al alumno en el desenvolvimiento de sus acciones. La ventaja de este tipo de aprendizaje es que el estudiante puede aprender a través de un guía y un procedimiento preestablecido, con lo que sus acciones son monitorizadas y se ofrece una gran yuda al usuario. En cambio, el laboratorio virtual que se va a extender en este trabajo sí que puede proporcionar estas ayudas al usuario, por lo que se puede considerar una herramienta educativa más avanzada que las que se han presentado en este apartado.

Capítulo 3

Planteamiento del Problema

En este capítulo se describen algunos conceptos básicos sobre Biotecnología, la estructura de un laboratorio de Biotecnología real y algunos de los procesos que se llevan a cabo dentro de él. Además, se resume el trabajo previo a este documento en el que se diseñó y construyó de manera casi completa el "Laboratorio Virtual de Biotecnología Agroforestal"[1] de la Universidad Politécnica de Madrid. Posteriormente, se describirá el proceso de Transformación Bacteriana que será el motivo de realización de este trabajo. Por último, se describirán aquellos problemas relacionados con la usabilidad del visor Firestorm que se identificaron en las primeras pruebas con profesores y, que serán resueltos en el siguiente capítulo.

3.1. Conceptos Básicos sobre Biotecnología

La Biotecnología [Clark and Pazdernik, 2009] es un campo de la ciencia que hace uso de organismos vivos dentro de los procesos industriales, especialmente en la agricultura, procesamiento de alimentos y en medicina. En estos tratamientos, algunas bacterias tienen la ventaja de ser fácilmente manipulables, ya que pueden reproducirse *in vitro* para ser estudiadas y modificadas genéticamente. Originalmente el concepto de biotecnología estaba dentro del campo de la ingeniería bioquímica. Sin embargo, en los últimos años ha adquirido un significado más amplio y, actualmente, se puede estudiar en otros campos tales como los alimentos, la biología molecular, entre otros.

[1] Vídeos de Demostración del Mundo Virtual de Biotecnología Agroforestal. http://serviciosgate.upm.es/laboratoriosvirtuales/laboratorios/biotecnologiaAgroforestal

3.1.1. Manipulación de la información genética

Dentro de las principales estrategias utilizadas en biotecnología destaca la genética directa, en la que se incrementa la expresión de un gen específico, produciendo una sobre-expresión. Por otro lado, la genética reversa o inversa produce la anulación de la expresión del gen de interés, el gen se interrumpe, modifica o se silencia para que no produzca la proteína.

3.1.2. Clonado de genes

Vector de Clonación. Es una molécula de ADN con información genética que es transferida al organismo huésped, replicándose en un número alto de copias dentro del genoma en el que se ha introducido. Los principales tipos de vectores de clonación son el plásmido, el fago (*Phage*), el cósmido, el BAC (*Bacterial Artificial Chromosomes*) y el YAC (*Yeast Artificial Chromosomes*).

Para esta práctica, el más relevante es el ADN plasmídico o plásmido. El ADN Plasmídico es un elemento extra cromosómico, generalmente con forma circular, que se localiza en el citoplasma de una bacteria. Esta cadena de ADN es idependiente de la que que se encuentra en el núcleo bacterial y es capaz de replicarse autónomamente. El pDNA es propio de los microorganismos procariotas, aunque algunas veces es posible encontrarlo en eucariotas. Los plásmidos son capaces de transferirse a otro unicelular y de replicarse dentro del organismo huésped.

Gen marcador de resistencia a antibióticos. Es un gen específico que concede resistencia a un antibiótico, además, acompaña al gen de interés en el vector de clonación. Este gen permite que el organismo pueda sobrevivir en un medio con presencia de dicho antibiótico, además de identificar cuál de ellos es el organismo resistente.

3.1.3. PCR (Reacción en Cadena de la Polimerasa)

La PCR es una reacción en cadena de la polimerasa que permite la amplificación del gen de interés. Está catalizada por la enzima Taq polimerasa (derivada de la bacteria termófila *Thermus aquaticus*) cuya temperatura óptima de actuación es de 72°C.

3.1.4. Transformación de especies vegetales

Mediante este método se modifica el genoma (información genética) de una célula vegetal.

***Agrobacterium tumefaciens* :** Es una proteobacteria[2] de la familia *Rhizobaceae* y una especie del género *Agrobacterium*, un agente etiológico que produce enfermedades en las plantas debido a su naturaleza parásita, generando tumores en sus tallos, raíces o en sus intersecciones. La *Agrobacterium tumefaciens* infecta a más de 90 millones de plantas de tipo dicotiledóneas y, en una menor cantidad, a las monocotiledóneas, lo que resulta perjudicial para el sector agrónomo. La infección se produce cuando el plásmido de la bacteria infecta las células vegetales, lo que conlleva una alteración en la estructura genética. Los genes ubicados en el ADN-T pueden ser reemplazados por cualquier otra secuencia de ADN, lo que resulta ser beneficioso para la investigación en el campo de la Biotecnología[Wood et al., 2001]. En la figura 3.1[3] se muestra la infección de la variedad *Agrobacterium vitis Ophel* en un árbol de tipo leñoso; nótese cómo los tumores crecen en el tallo de la planta.

Figura 3.1: Detalle de un árbol infectado con *Agrobacterium tumefaciens*.

En este laboratorio de Biotecnología, el *Agrobacterium tumefaciens* ha sido modificado genéticamente, eliminando la información que causa el tumor en la planta, pero manteniendo su capacidad de insertarse en el genoma, lo que permite su uso en biotecnología para la obtención de organismos modificados genéticamente.

Medio de Cultivo : Una de las ventajas del laboratorio de Biotecnología es que se pueden cultivar y observar el crecimiento de microorganismos. Para que esto sea posible

[2]Tipo de bacteria que incluye a los agentes patógenos, tales como *"Escherichia, Salmonella, Vibrio, Helicobacter, Neisseria gonorrhoeae, la Burkholderia glumae"*[Wikimedia, 2013], entre otros.

[3]Imagen extraída de la página web de Agromática. http://www.agromatica.es/agrobacterium-tumefaciens/

se requiere de un medio de cultivo especializado para cada tipo, cada uno se caracteriza por presentar ciertas condiciones como: nutrientes, temperatura, grado de humedad, nivel de oxígeno, grado de pH (alcalinidad o acidez) y esterilidad. Para el crecimiento de las bacterias patógenas como la *Agrobacterium tumefaciens*, es muy común el uso del Agar, una sustancia solidificante que entra en estado líquido a los 100°C y que se solidifica a los 40°C. Cada uno de los medios de cultivo se encuentran enriquecidos con hidratos de carbono, sueros, etc.; además, existen diversos indicadores que permiten conocer el estado del pH del medio, de alguna manera, para comprobar si las condiciones del medio son realmente eficientes para el desarrollo de los microorganismos. En la figura 3.2[4] se muestra el Agar como medio de cultivo colocado en una placa Petri durante un experimento.

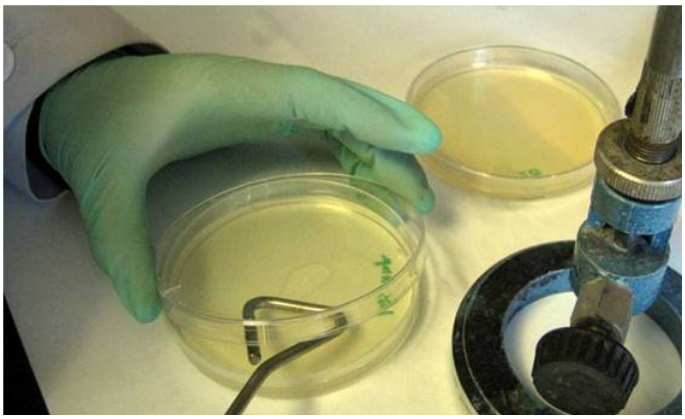

Figura 3.2: Agar como medio de cultivo presentado en placas Petri.

Además del Agar, existen otros tipos de medio de cultivo muy utilizados como el *Murashige and Skook* (mezcla de vitaminas, minerales y otros nutrientes), la sacarosa (azúcar de tipo disacárido utilizado como fuente principal de carbono para el metabolismo de la planta) y el ácido indolacético (IAA, es una hormona que induce la formación de raíces).

3.1.5. Cultivo *in vitro* de especies árboreas y vegetales

Micropropagación

Es una técnica de regeneración de material vegetal. En esta práctica se generan nuevos individuos a partir de un árbol de chopo de tres meses, que han crecido en un entorno

[4]Imagen extraída de la página web "Prácticas Online de Microbiología para Farmaceúticos" de la Universidad de Granada. http://www.pomif.com/pages/practicas/bacteriologia/transformaciondeecoli/texto-transfecoli/transformacion-metodo-y-resultados

apropiado dentro de un fitotrón o cámara visitable de crecimiento de plantas. El cultivo *in vitro* permite una mayor rapidez en el crecimiento de las plantas bajo condiciones en las que se tiene un control sobre el tiempo, temperatura, humedad, luz, etc; además, posibilitar tratamientos específicos, transformación de tejidos y células.

3.1.6. Laboratorio de Biotecnología real

Dentro de un laboratorio de Biotecnología, se realizan diversos procedimientos que requieren de un entorno adecuado con la maquinaria, material biológico, material bioquímico e instrumentos de laboratorio necesarios. Por tanto, se necesita de un laboratorio de Biotecnología en el que se puedan llevar a cabo todos estos experimentos. La regla básica [Chawla, 2002] en el diseño de este tipo de entornos es el mantenimiento del orden y de la limpieza dado que todas las operaciones requieren de entornos asépticos.

Según [Chawla, 2002], un laboratorio de Biotecnología debe disponer de:

- **Laboratorio y área de preparación**
  Es el área principal del laboratorio en la que se realizan la mayoría de las actividades. Incluye un área de preparación y de autoclave. Además, es común que cuenten con el equipamiento básico como refrigeradores, congeladores, pH-metro, hornos, balanzas microscópicas, entre otros.

- **Área de transferencia**
  Algunas técnicas de biotecnología requieren de un ambiente estéril, libre de microorganismos. La cabina de flujo es la maquinaria más utilizada para estos casos y funciona de tal manera que una corriente de aire es liberada desde el interior hacia el exterior.

- **Área de cultivo o fitotrón**
  Los vegetales necesitan ser incubadas bajo ciertas condiciones controladas como la temperatura, iluminación, periodos de luz, humedad y circulación de aire.

- **Invernadero**
  Esta área es requerida para el crecimiento de plantas modificadas hasta su madurez, durante el cual son aclimatadas antes de ser trasladadas a campos reales.

Además de las áreas mencionadas anteriormente, en el laboratorio de Biotecnología Agroforestal de la UPM se cuenta [Riofrío Luzcano, 2012] con una sala de computadoras en la que se dispone de software necesario para la realización de cálculos y otras actividades relacionadas con la investigación. Los laboratorios son divididos por niveles de seguridad,

desde P1 hasta P4; el nivel inferior requiere de menos protección para el especialista y puede que sólo utilice guantes de goma; mientras que en el área P4, la utilización de trajes protectores es un requisito obligatorio.

Algunos de los instrumentos típicos en un laboratorio de Biotecnología y que son utilizados en la Transformación Bacteriana son[Riofrío Luzcano, 2012]:

- **Poyata**: Mesa de laboratorio o meseta, es un lugar en el que se pueden realizar ciertas operaciones de un experimento, como mezclas de sustancias o revisiones de algunos documentos o material orientativo. Debe disponer de una resistencia mecánica y química para que puede ser fácilmente lavado y descontaminado.
- **Autoclave**. Maquinaria que permite la eliminación de microorganismos de la sustancia o material que se coloca dentro de él.
- **Cabina de Flujo Laminar**: Mantiene una estado estéril para poder trabajar con cultivos de células vegetales o tejidos.
- **Tubo *Eppendorf***: Es un tubo de polipropileno utilizado para el autoclavado y la centrifugación. Puede soportar temperaturas de hasta -20°C [Wikipedia, 2013b].
- **Placa Petri**: Es un recipiente redondo de vidrio utilizado en el cultivo de plantas y microorganismos.
- **Aparato de Electroporación**: Es un aparato que tiene la capacidad de abrir los poros de las paredes bacterianas a través de una diferencia de voltajes.
- **Cubeta de Electroporación**: Especie de tubo de plástico con electrodos de aluminio y tapa azul. Es utilizado en el proceso de electroporación[Wikipedia, 2013a].
- **Pipeta**: Es un instrumento utilizado para la toma de líquidos en cantidades pequeñas y con bastante precisión. En esta práctica se utilizan las pipetas amarillas y azules que varían en la cantidad de líquido que pueden tomar.

3.1.7. Práctica: Transformación genética de un chopo para dotarle de resistencia a ciertos hongos

En este trabajo se continuará con la práctica de transformar genéticamente un chopo para mejorar su resistencia a ciertos hongos. Para cumplir con este objetivo, la práctica está dividida en una serie de fases que deben de seguirse de forma estricta. Es fundamental

que en la repetición de este procedimiento se puedan obtener resultados idénticos, ya que se hace aplica el método científico y se debe confirmar la hipótesis planteada.

Las fases que se deben de seguir en esta práctica son las siguientes:

1. Micropropagación de material vegetal. A partir de una planta no modificada, mediante la clonación, se obtienen varias genéticamente iguales.

2. PCR. En esta fase se amplifica el gen de interés de un ADN molde de chopo.

3. Ligación. El gen amplificado de la fase anterior se introduce en el plásmido.

4. Transformación bacteriana. El plásmido es insertado en la bacteria *Agrobacterium tumefaciens*.

5. Transformación vegetal. En esta fase se obtienen los árboles modificados genéticamente. Se inserta el gen de interés (amplificado) en la célula vegetal mediante la infección de la bacteria *Agrobacterium tumefaciens*.

6. Inoculación de tejido vegetal. En esta fase se comprueba la resistencia de la planta a la infección. Los hongos son inoculados a una planta silvestre y a una planta modificada, verificando que la manipulación genética ha sido realizada con éxito.

De las fases mencionadas, en [Riofrío Luzcano, 2012] se implementó la "Micropropagación de material vegetal", posteriormente, el mismo autor continuó con la fase de la "PCR", finalmente, en la tesis [Pedraza, 2013] se concluyó con la fase de la "Ligación". En este trabajo se seguirá con el diseño e implementación de la "Transformación Bacteriana", en el que se configurarán las acciones correspondientes a la fase y a todas las validaciones que éstas involucran.

3.2. Antecedentes

En esta sección se describirá el laboratorio virtual de Biotecnología desarrollado e implementado en [Riofrío Luzcano, 2012], además, se mencionarán las fases de la práctica que fueron implementadas en trabajos anteriores.

3.2.1. Laboratorio virtual de Biotecnología

El laboratorio de Biotecnología había sido previamente desarrollado tal como se menciona en [Riofrío Luzcano, 2012]; este mundo virtual fue construido e instalado en la plataforma OpenSim sobre un sistema operativo Microsoft Windows 7 y con una base de

datos MySQL 5.5. Para poblar el laboratorio con instrumentos y maquinarias se crearon objetos 3D. Del modelado de estos objetos se encargó el propio autor y la empresa *Novatierra*[5]. Adicionalmente, otros objetos fueron descargados de la página web *OpenSim Creations*[6].

La construcción de las salas del laboratorio estuvo acorde a lo dispuesto en la sección 3.1.6, si bien sólo se han modelado aquellas salas que se utilizan en la práctica virtual implementada en [Riofrío Luzcano, 2012]. El conjunto total de salas conformaba un edificio que se describe a continuación.

- **Vestíbulo**. Sala de bienvenida utilizada como sala de recreo para los estudiantes y la selección de las prácticas.
- **Patio**. Área externa al edificio, también utilizada como sala de recreo para los estudiantes.
- **Sala Principal**. Área reservada para que los alumnos dispongan de un espacio de trabajo (poyata) en el que puedan realizar algunos procedimientos comunes.
- **Sala de Cabinas de Flujo**. Área en la que se encuentran las cabinas de flujo laminares.
- **Fitotrón**. Área diseñada para el crecimiento de las plantas en condiciones controladas.
- **Sala de autoclavado**. Sala en la que se encuentran los autoclaves para la esterilización y limpieza de los instrumentos.

La disposición de las salas se muestra en la figura 3.3. Como se puede apreciar, cada una de las salas está separada una de otra y el avatar puede navegar entre ellas con ayuda de letreros (figura 3.4) que le indican dónde se encuentra cada una de ellas.

[5]http://www.novatierra.com
[6]http://opensim-creations.com

Figura 3.3: laboratorio de Biotecnología virtual.

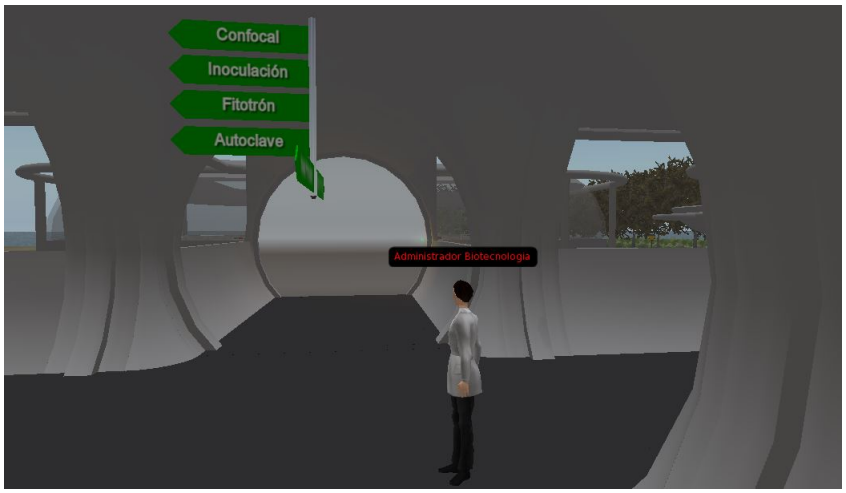

Figura 3.4: Señalización en el laboratorio de Biotecnología virtual.

3.2.2. Práctica virtual ya implementada

Como se había mencionado anteriormente, las tres primeras fases de la práctica ya han sido implementadas en [Riofrío Luzcano, 2012] y en [Pedraza, 2013]. A continuación, se describirá brevemente lo desarrollado en cada una de ellas.

1 **Micropropagación de material vegetal** [Riofrío Luzcano, 2012]. Esta fase fue desarrollada con dos propósitos: verificar que el tutor automático valide correctamente las acciones configuradas y empezar el desarrollo de la práctica de laboratorio. Este proceso fue dividido en dos subflujos: Preparación de medio de enraizamiento y la Micropropagación de material vegetal propiamente dicha. En el primer subflujo se realiza la preparación del medio de cultivo en el que las plantas crecerán posteriormente; y en la segunda, una planta cortada en pequeños fragmentos se coloca en un tubo de ensayo que contiene el medio de cultivo, se lleva este tubo al fitotrón y se espera su crecimiento.

2 **PCR**. Esta fase fue desarrollada para amplificar el gen de interés mediante la mezcla del ADN molde de chopo con una serie de sustancias dentro de un tubo *Eppendorf*. Posteriormente, esta mezcla es colocada en el aparato PCR para amplificar el gen de interés y seguidamente el fragmento amplificado es colocado en un frigorífico a -20°C.

3 **Ligación** [Pedraza, 2013]. En esta fase se hace uso del termobloque para introducir el gen amplificado en la fase anterior en el plásmido vegetal. En este procedimiento

se realiza una mezcla del ADN plasmídico, el ADN amplificado por la PCR y otras sustancias dentro de un tubo *Eppendorf*. Posteriormente, se coloca el tubo dentro de un termobloque a 16°C y una vez terminado el proceso de ligación, se guarda el plásmido en un frigorífico a -20°C.

3.3. Fase de la Práctica a implementar: Transformación Bacteriana

Dentro de los procedimientos que se pueden realizar dentro del laboratorio de Biotecnología, la Transformación Bacteriana tiene como objetivo el insertar el plásmido vegetal que contiene el gen de interés y que fue aislado en la fase de la Ligación, dentro de la bacteria *Agrobacterium tumefaciens*, que es una bacteria capaz de infectar la planta y modificar su genoma introduciendo el gen mencionado.

Este plásmido vegetal ingresa en la bacteria a través de su membrana celular, mediante el proceso de electroporación, en el que debido a una diferencia de voltajes, los poros de la membrana celular se expanden para que el plásmido vegetal pueda ingresar en el citoplasma y combinarse con el plásmido bacterial.

3.3.1. Descripción del proceso

Para lograr la Transformación Bacteriana se siguen ciertos pasos que deben de considerarse rigurosos y subsecuentes, tal como se mencionan a continuación:

1. Las bacterias *Agrobacterium tumefaciens* se encuentran en un frigorífico a -80°C ubicado en la sala principal, se colocan dentro de una bandeja de hielo y se descongelan por, aproximadamente, 15 minutos.

2. El ADN plasmídico que se encuentra en un frigorífico a -20°C ubicado en la sala principal, se coloca dentro de la bandeja.

3. En la cabina de flujo se realiza la mezcla del *Agrobacterium tumefaciens* y el ADN plasmídico dentro de una cubeta de electroporación con ayuda de una pipeta amarilla y azul, respectivamente.

4. Dentro de la misma cabina de flujo se introduce la cubeta en un aparato de electroporación y, a continuación, se enciende la fuente eléctrica para iniciar la electroporación. Se producirá un choque eléctrico con el que las paredes bacterianas se abren y el plásmido ingresa al microorganismo.

5. Dentro de la cabina de flujo se traspasa el resultado de la electroporación en un tubo *Eppendorf* y, posteriormente, es colocado en el horno a 37℃ ubicado en la sala principal durante una noche para que las bacterias entren en recuperación a través de un proceso de agitación.

6. Dirigirse al Autoclave a por una botella con medio de cultivo.

7. Luego de haber terminado la agitación en el horno, se debe regresar a la cabina de flujo para colocar el medio de cultivo y la mezcla del tubo *Eppendorf* en una placa Petri.

8. A la placa Petri se le adicionan antibióticos ubicados en el frigorífico a -20˚ubicado en la sala principal con el fin de eliminar aquellas bacterias que no hayan adquirido el plásmido, y por tanto no sean resistentes a los antibióticos.

9. La placa Petri es colocada nuevamente en el horno a 37°C ubicado en la sala principal y se espera una noche para que las bacterias se multipliquen y aparezcan las colonias.

10. Una vez terminada la segunda agitación, se regresa nuevamente a la cabina de flujo y con ayuda de una pipeta amarilla, en un falcon se coloca una de las colonias que ha sobrevivido a los antibióticos y se lleva nuevamente al horno a 37°C ubicado en la sala principal por otra noche más.

11. Finalmente, se retira el falcon, se regresa a la cabina de flujo para colocarlo en una rejilla y concluye el proceso de Transformación Bacteriana.

3.4. Problemas de Usabilidad con el visor Firestorm

La forma de interactuar de una persona a través de su avatar con el mundo virtual es un factor importante y decisivo que puede afectar la aprobación por parte del usuario de un sistema desarrollado dentro de un entorno virtual de aprendizaje (VLE, *Virtual Learning Environment*).

Si el usuario nota cierta dificultad al navegar por el mundo, además de las dificultades propias de las prácticas de laboratorio, se puede producir cierto rechazo de su parte. Por esa razón, es preciso brindarle todas las facilidades para que el flujo de la práctica no se vea obstaculizado, es decir, mejorar la usabilidad del visor con el cual se conectan al laboratorio de Biotecnología.

Dentro de los visores disponibles para conectarse a OpenSim se encuentran *Firestorm* [Inc., 2013], *Kokua* [Imprudence, 2013], *Singularity* [Singularity, 2013], *Radegast Metaverse* [Radegast, 2013], entre otros; de los mencionados se optó por elegir *Firestorm*, un proyecto de "*The Phoenix Firestorm, Inc.*" debido a que se tenía acceso al código fuente y era uno de los que presentaba un mejor rendimiento frente a otras alternativas. Asimismo, como consecuencia de la disponibilidad del código fuente, se pudieron realizar todas las modificaciones necesarias para mejorar la usabilidad.

Luego de haber implementado las dos primeras fases de la práctica (Micropropagación de material vegetal y PCR), se inició una fase de pruebas con profesores e investigadores, como resultado de estas pruebas, se revelaron ciertos problemas ligados a la usabilidad del visor. Los problemas encontrados se exponen a continuación:

3.4.1. Bloqueo a la práctica

Existen acciones en las que se entregan objetos al avatar; posteriormente, el visor muestra un mensaje de confirmación en el que se solicita permiso para agregarlo al inventario. Forzosamente el usuario debe de presionar el botón "Guardar", ya que si presiona el botón "Descartar" o "Bloquear al propietario", no se le entrega el objeto y no puede continuar con la práctica a causa de que las siguientes acciones requieren que el avatar posea el mencionado objeto. Como se puede visualizar en la figura 3.5, el mensaje que aparece en el cuadro celeste pide permiso al avatar para que pueda ser agregado al inventario. La modificación que se va a realizar consistirá en eliminar esta confirmación innecesaria, que conlleva el riesgo que ya se ha explicado.

Figura 3.5: Mensaje de confirmación para agregar objeto a inventario.

3.4.2. Mensajes innecesarios

Un mensaje que se mostraba después de haber entregado un objeto al inventario contenía información del objeto (un identificador de tipo GUID[7], el ID del objeto y su posición en el mundo y otros datos adicionales); estos mensajes, si bien no se pueden considerar bloqueantes como los del punto anterior, muestran información innecesaria al usuario y obligan a realizar una acción más, ya que se tenían que cerrar las ventanas de los mensajes, lo cual resultaba incómodo. Otro mensaje innecesario aparece cuando se sitúa el puntero encima de un objeto. Ambos mensajes se presentan como se muestra en las figuras 3.6 y 3.7, respectivamente. En consecuencia, se decidió eliminar todos estos mensajes, que solo podrían servir para crear confusión en el usuario.

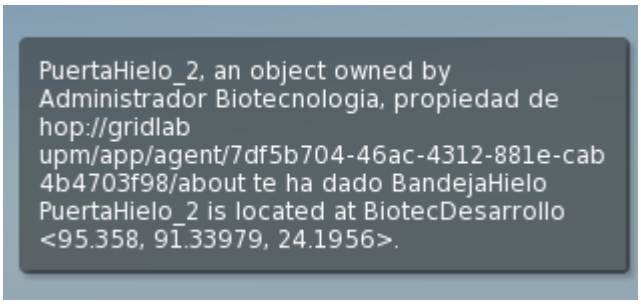

Figura 3.6: Mensajes al agregar objeto a inventario.

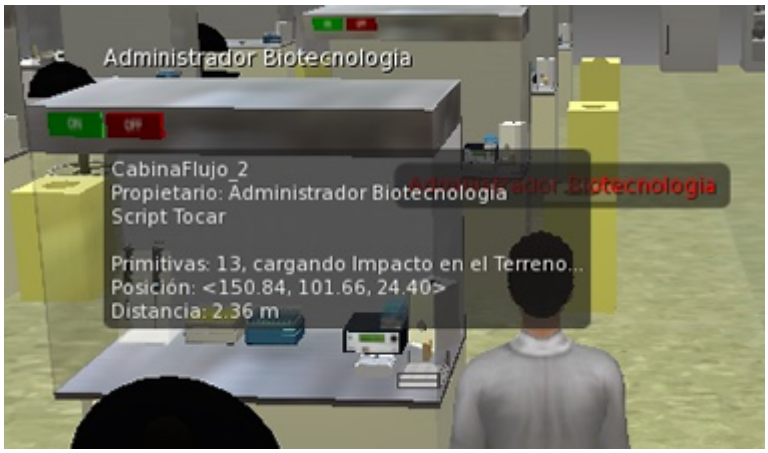

Figura 3.7: Mensajes tooltip de objetos.

3.4.3. Soltar objeto adjuntado con mayor facilidad

Adicionalmente, dentro de la práctica existen algunos pasos en los que el avatar debe de soltar un objeto que lleva en la mano. Esto traía consigo cierto nivel de dificultad para

[7] *Global Unique Identifier*. Es un identificador único y un número de 128 bits que es utilizado en sistemas operativos Microsoft Windows, por ejemplo : "25bdf4c3-6aa5-410f-9aeb-892ebdf830c0".

un usuario aún inexperto en la interacción con el mundo virtual, ya que en algunos casos, el avatar se mostraba detrás del objeto adjuntado, por lo que se debía de girar la cámara con los movimientos del ratón para tener un plano en el que el objeto podía ser cliqueado. Sin embargo, algunas veces, esta rotación de la cámara resulta difícil y si no se realiza correctamente, el usuario puede perder el foco del avatar, del objeto o hasta de la misma zona donde se encuentra. La imagen izquierda de la figura 3.8 es la posición en la que el avatar permanece normalmente en la vista de la cámara por defecto; mientras que la de la derecha es la posición deseada para poder hacer clic sobre el objeto adjuntado. Con el fin de facilitar esta operación al usuario, se decidió añadir un botón visible en todo momento que permita soltar el objeto que el avatar lleve en la mano.

Figura 3.8: Cambio de posición para poder soltar un objeto adjuntado.

3.4.4. Configuraciones por defecto

Como configuraciones por defecto, dentro de Firestorm se encuentran los nombres de las redes disponibles, es decir, de aquellos servidores en los que se ejecuta OpenSim y las regiones por defecto en aquellos mundos. Para el objetivo de la práctica de laboratorio es irrelevante listar estos servidores de OpenSim. Además, dado que el servidor del laboratorio de Biotecnología no se encuentra en esa lista, esto representaba un cierto inconveniente ya que se tiene que configurar manualmente en cada cliente instalado, agregando dicho servidor a la lista. Por tanto, se decidió dejar en la lista únicamente el servidor del laboratorio virtual fijando como región por defecto la que contiene dicho laboratorio.

Además, en cuanto a la configuración gráfica existen diversos estilos o temas, en donde cada uno ofrece una apariencia distinta y una configuración singular de colores y

formas aplicables al visor. Adicionalmente, cada configuración tiene su propio diseño del menú contextual para todos los elementos del mundo virtual, algunas configuraciones lo presentan verticalmente y otros, con forma de tarta; de igual manera, los mensajes de chat o los informativos varían en cuanto a transparencia y colores. De entre las configuraciones disponibles se escogió la más adecuada para el usuario.

Un problema de usabilidad respecto al menú contextual es que existen ciertas opciones que el usuario no va a utilizar en el desarrollo de la práctica de laboratorio (tales como *crear*, *comprar*, *tomar* o *pagar*), por lo que no deberían estar disponibles, ya que podrían crear confusión en un usuario inexperto. Además, la presentación de los ítems del menú también requiere cierto grado de atención ya que si el menú es de tipo tarta, algunas opciones que se utilizan con frecuencia no se muestran en una primera vista, por lo que el usuario tiene que hacer clic en el botón *"Más"* para visualizar las demás acciones lo que, lógicamente, implica un paso más. La figura 3.9 muestra el menú contextual de un objeto que se tiene adjuntado. En consecuencia, se eligió la configuración que utiliza un menú de tipo lista y se eliminaron de esa lista todas aquellas opciones o comandos que no se utilizan en la práctica de biotecnología.

Figura 3.9: Menú contextual por defecto de los objetos.

3.4.5. Claridad de los mensajes

Como punto final, el laboratorio de Biotecnología ha sido construido con una luminosidad similar a uno real, tan solo los suelos o las puertas presentan tonalidades oscuras que ocupan gran espacio. Considerando esto, los mensajes del chat o similares deberían resaltarse para que puedan ser leídos sin ninguna dificultad, para lo cual los colores del fondo o colores de letra en los mensajes no deberían mezclarse con colores similares de los objetos del laboratorio. En la figura 3.10 se puede apreciar la transparencia de los

mensajes en la configuración por defecto del visor y la dificultad de su lectura. Para contrarrestar este problema se vio la necesidad de modificar los valores de los atributos que incurrían en este tipo de dificultades. Estas características se encontraban definidas en unos ficheros de configuración del visor.

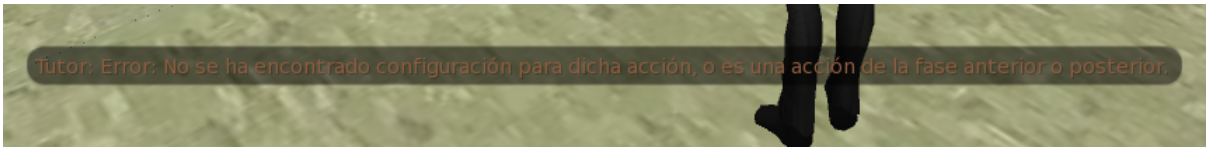

Figura 3.10: Transparencia de los mensajes.

Capítulo 4

Solución Adoptada

En este capítulo se describirá cómo está estructurado el tutor automático y su funcionamiento. Además, se explica el diseño de la Transformación Bacteriana, el cual se documenta mediante distintos modelos que muestran la estructura del sistema y su funcionamiento desde distintos puntos de vista. Finalmente, se explicarán todos los cambios realizados en el visor Firestorm, correspondientes a los problemas encontrados en la sección anterior.

4.1. Diseño e Implementación de la Transformación Bacteriana

En esta sección se describirá el funcionamiento del tutor automático, así como el procedimiento definido en su archivo de configuración para la Transformación Bacteriana. Para una representación gráfica, se utilizó una notación que se aplica al protocolo de esta fase. Además, se mencionarán los objetos 3D que fueron construidos y la relación de éstos con los ya existentes. Finalmente, se presentan los eventos, mensajes y actividades de todos los objetos partícipes de esta fase y los diagramas de secuencia que forman parte de la solución.

4.1.1. Descripción del tutor automático

Según lo descrito en [Riofrío Luzcano, 2012], para validar las acciones que se realizan durante la práctica, se construyó una primitiva llamada “tutor”. Este objeto es invisible para los avatares y tiene la funcionalidad de recibir peticiones a través del canal número 1 y devolver una respuesta válida o inválida a través del canal número 2.

Para almacenar los estados de los usuarios en el transcurso de la práctica, se tienen los siguientes archivos de texto plano (*notecard*) dentro del inventario del tutor:

- **Archivo de configuración**
  En este archivo se almacena la información de las fases, los mensajes de validación, las dependencias y algunos indicadores. Éste es el archivo principal del tutor en el que están registradas todas las acciones válidas e inválidas que el usuario puede realizar a lo largo de la práctica. Su estructura se explicará con más detalle en la sección *"Configuración del tutor"*.

- **Archivo de fases**
  En este archivo se almacenan los identificadores GUID de los usuarios con las correspondientes fases en las que se encuentran. Cada línea del archivo tiene 3 valores que están separados por una barra vertical (|), tal como se muestra a continuación:

  4a97fe61-6b5d-4baa-8066-1ce13949f27d|f 0|1

  El primer valor corresponde al GUID del avatar; el segundo, se refiere a la fase en la que se encuentra; y la tercera se corresponde con la práctica que seleccionó inicialmente.

- **Archivo de bloqueos**
  En este archivo se tiene un listado de objetos que pueden ser bloqueados por un avatar. Una vez bloqueado, ningún otro usuario puedo hacer uso de él. Los objetos se bloquean mediante un indicador en el archivo **ConfiguracionPopulus**. Cada lńea está separada por una barra vertical (|), tal como se muestra a continuación:

  CabinaFlujo_2|f2dfc0ce-8b77-448f-9793-7231fcaf2d70

  El primer valor corresponde al objeto bloqueado y el segundo, al GUID del avatar. En caso de no tener el GUID, el objeto se encuentra desbloqueado y disponible para cualquier avatar. Una vez que el objeto se encuentra bloqueado, puede ser desbloqueado, a través del mismo indicador, al realizar una acción posterior.

- **Archivo de usuario**
  Este archivo se crea cada vez que un usuario inicia una práctica y se actualiza conforme el usuario avanza por las fases. Además, se van agregando líneas que corresponden a las acciones realizadas por el usuario.

Los valores de la línea separados por la barra vertical (|) se corresponden con el código secuencial de la acción, el nombre de la acción, el tiempo que se tardó en ejecutar la acción (sólo para acciones temporales), los códigos de las dependencias insatisfechas, los códigos de las acciones incompatibles que estaban presentes y la fecha y hora del registro de la acción, respectivamente. En la siguiente línea se muestra una acción realizada correctamente, que no tiene ni dependencias ni acciones incompatibles:

f4t1|encendercabina| | | |2013-06-21T10:19:14.0000000Z

Por otro lado, en la siguiente línea se observa que la acción "soltarbandejapoyata" tiene asociada la dependencia "f4t7" que no ha sido realizada, mientras que la accion "f4t27" representa una de acción incompatible conflictiva:

f4t29|soltarbandejapoyata| |[f4t7]|[f4t27]|2013-06-21T10:19:14.0000000Z

Configuración del tutor

El archivo de configuración del tutor contiene todas aquellas acciones que pueden ser realizadas por el usuario. Estas acciones tienen asociados mensajes que el usuario deberá utilizar como guía para completar la práctica. La estructura de cada acción del tutor y sus atributos se encuentra separada por una barra vertical (|). Los atributos de cada línea del tutor se describen en el cuadro 4.1.

Atributo	**Descripción**
Fase/Tarea	Identificador de acción.
Código Acción	Nombre de la acción.
(1)/(0)-MensajeOK	Mensaje válido(1) y oculto(0).
MensajeAccionSiguiente	Mensaje de siguiente acción.
Dependencias	Acciones de las que depende.
(1)/(0)-MensajesErrorDependencias	Mensaje de error en caso de que no se cumplan todas las dependencias.
(1)/(0)-MensajesErrorOrdenDepend	Mensajes de error en caso de que no se respete el orden de las dependencias.
Incompatibilidades	Acciones incompatibles.
(1)/(0)-MensajeErrorIncompatibillidad	Mensaje de error por las incompatibilidades.
Bloquea(1)Desbloquea(0)	Indicador de bloqueo de objeto.
TiempoMaximo	Tiempo máximo para la ejecución de la acción.
TiempoMinimo	Tiempo mínimo para la ejecución de la acción.
ValidarAccionYaRealizada	Indicador para validar la repetición de la acción.
ValidarErroresFase	Indicador para validar los errores en la fase.
CambiodeFase	Indica que la acción es la última de una fase.

Cuadro 4.1: Estructura del tutor automático

Los atributos que van antepuestos por (1) ó (0), indican que el mensaje asociado al valor se muestra u oculta en la ventana de chat, respectivamente. Una trama de ejemplo que contiene toda la estructura del cuadro 4.1, se muestra a continuación:

f4t1|encendercabina|1-Cabina de flujo encendida.|Espera el tiempo necesario para que se esterilice el aire.| | | | | |1 |0 |0| | | 1| |1

En el ejemplo anterior, cuando el usuario presiona el botón “Encendido” de la Cabina de Flujo, se envía la acción “encendercabina” al tutor y éste responde con el mensaje “encendercabina” por el canal número 2. Además, envía, a través de la ventana de chat del usuario, el mensaje “Cabina de flujo encendida.” y luego el mensaje, “Espera el tiempo necesario para que se esterilice el aire.”.

Por otro lado, una trama con la siguiente estructura, representa una acción cuya dependencia es la acción “[f5t1]” y que mostrará el mensaje “No se ha cerrado el electro-

porador."en caso no se haya realizado previamente.

f5t2|encenderfuenteelectrop|1-Se ha concluído la electroporación.|Ahora toque el soporte para electroporar.|[f5t1]|1-No se ha cerrado el electroporador.| | | | | |10| | | 1| |

Para el desarrollo de la Transformación Bacteriana, se agregaron al archivo de configuración del tutor, todas las acciones necesarias para cumplir con lo señalado en la sección 3.3.1 del capítulo anterior.

Funcionamiento del tutor

El tutor automático escucha siempre por el canal número 1 y recibe, por parte de los objetos, una trama formada por una estructura que se describe en el cuadro 4.2. De igual manera que en los notecards, cada campo está delimitado por barras verticales (|).

Atributo	**Descripción**
Operación	Indica la función de gestión.
GUID del avatar	Identificador del avatar.
Acción	Acción del tutor.
Nombre del objeto	Nombre del objeto que contiene el *script*.

Cuadro 4.2: Estructura del mensaje enviado al tutor.

Dentro de las operaciones que puede recibir el tutor, se encuentran "crearLibro", "borrarFase", "validar" y "borrar". Las dos primeras son acciones predeterminadas del tutor; mediante "crearLibro" se crea el "Archivo de usuario"con el nombre del avatar que está iniciando la práctica; y mediante "borrarFase", se eliminan todas las acciones de la última fase de la práctica, en el caso de que la fase sea la primera, también se borra el "Archivo de usuario".

Para el caso de la operación "validar", se envía la acción que debe ser validada por el tutor de acuerdo al "Archivo de configuración". Si la operación resulta ser válida, se agrega la acción en el "Archivo de usuario", indicando que la acción se ha ejecutado correctamente. Finalmente, la operación "borrar" sirve para eliminar una acción ya realizada del "Archivo de usuario", generalmente se usa para aquellas acciones que pueden ser repetidas durante una fase y para las acciones temporales con un tiempo máximo de ejecución. Para un mejor entendimiento del funcionamiento del tutor, se ha diseñado el diagrama de flujo mostrado en la figura 4.1. La descripción completa se encuentra en el Capítulo 4 de [Riofrío Luzcano, 2012].

Comúnmente, cuando se envía una operación de tipo "validar" desde un objeto, éste puede recibir tres tipos de respuestas por parte del tutor; cuando empieza con "1", la

acción se ha realizado correctamente y se mostrará un mensaje de chat con la siguiente acción que debe ser realizada; cuando empieza con “0”, la acción no se ha realizado correctamente y se mostrará un mensaje al usuario indicándole que no se ha realizado la acción; finalmente, cuando empieza con “-1”, la acción no ha sido ejecutada correctamente pero no se mostrará ningún mensaje al usuario y se le permitirá seguir con la práctica.

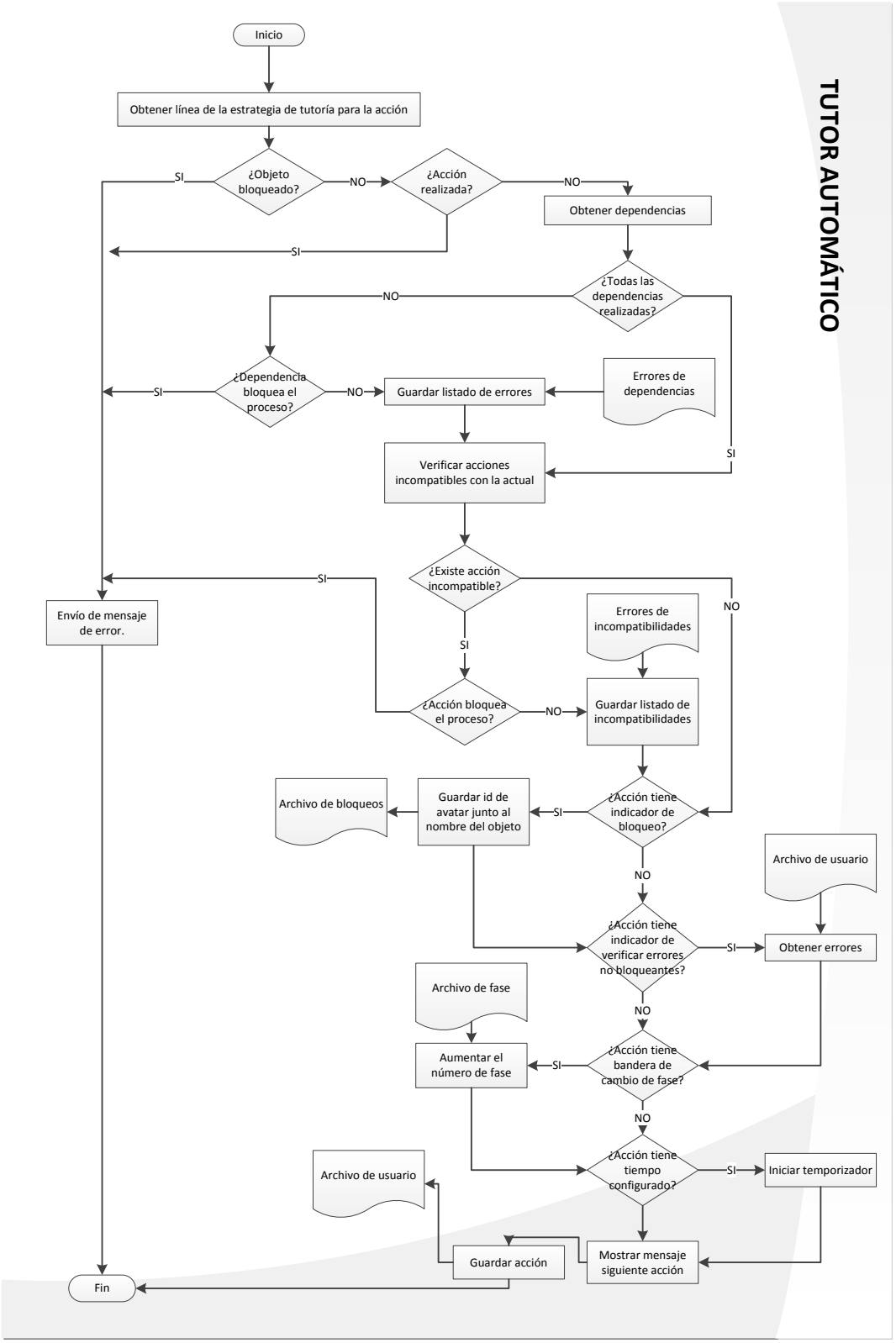

Figura 4.1: Diagrama de flujo del tutor automático.

4.1.2. Objetos 3D realizados

Dado que el Laboratorio Virtual de Biotecnología ya dispone de la mayoría del instrumental necesario para el desarrollo de la Transformación Bacteriana, el autor sólo tuvo que diseñar aquellos objetos que fueron necesarios para la Electroporación. Estos objetos 3D se muestran a continuación:

- **Cubeta de Electroporación** (Figura 4.2)
  Tubo de plástico que es utilizado en el proceso de electroporación y que, posteriormente, es colocado en el aparato de electroporación.
- **Aparato de Electroporación** (Figura 4.4)
  Aparato que aplica una descarga eléctrica a la cubeta de electroporación.
- **Fuente de Electroporación** (Figura 4.3)
  Proveedor de energía para el aparato de electroporación. El usuario debe de encenderla mediante un clic del ratón para que se inicie el proceso de electroporación.

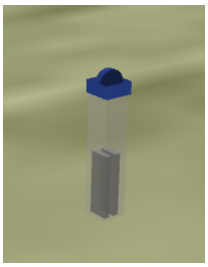

Figura 4.2: Cubeta de Electroporación.

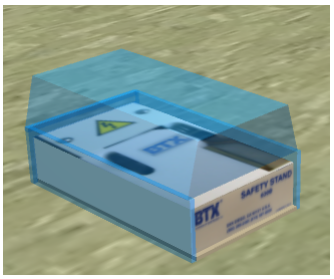

Figura 4.3: Aparato de Electroporación.

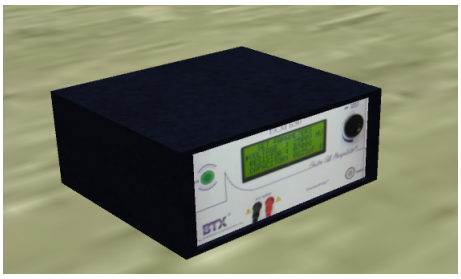

Figura 4.4: Fuente de Electroporación.

Los objetos mencionados fueron replicados en cada cabina de flujo, dado que se necesitaba de un entorno estéril para estas operaciones. Además, a cada objeto se le asignó un nombre en el que el sufijo se correspondía con el mismo número sufijo de la cabina de

flujo (por ejemplo, si la cabina de flujo se llamaba *CabinaFlujo_1*, el aparato de electroporación se llamaría *AparatoElectrop_1*); posteriormente, este nombre va a ser utilizado para bloquear la fuente y el aparato de electroporación en el "Archivo de bloqueos".

4.1.3. Notación del protocolo

Para tener una representación gráfica de las acciones del tutor, se ha utilizado una notación en la que se puedan visualizar con claridad las dependencias y las restricciones en el orden de su ejecución. Tal como se puede observar en la Figura 4.5, la notación está compuesta por dos tipos de componentes:

Acciones

Las acciones son representadas de manera similar a un proceso en un diagrama BPM. Mediante un rectángulo de color gris con esquinas redondeadas se denota una acción del usuario. Para denotar que un conjunto de acciones no tienen que ejecutarse en orden preestablecido, pero deben ejecutarse todas, éstas se encierran en un rectángulo de las mismas características pero con fondo blanco.

Flechas

Las relaciones entre las acciones se denotan mediante flechas; para cada tipo de relación existe una flecha distinta, tal como se describe a continuación:

- **Flecha normal** : Indica una dependencia de acciones. Se interpreta como "accion2" depende o requiere la ejecución previa de "accion1", siendo la "accion1" la que recibe la punta de la flecha.
- **Flecha bidireccional** : Indica que ambas acciones se anulan entre sí.
- **Flecha punta equis** : Indica que las acciones son incompatibles. Se interpreta como "accion1" es incompatible con "accion2", siendo la "accion2" la que tiene unida la equis de la flecha.

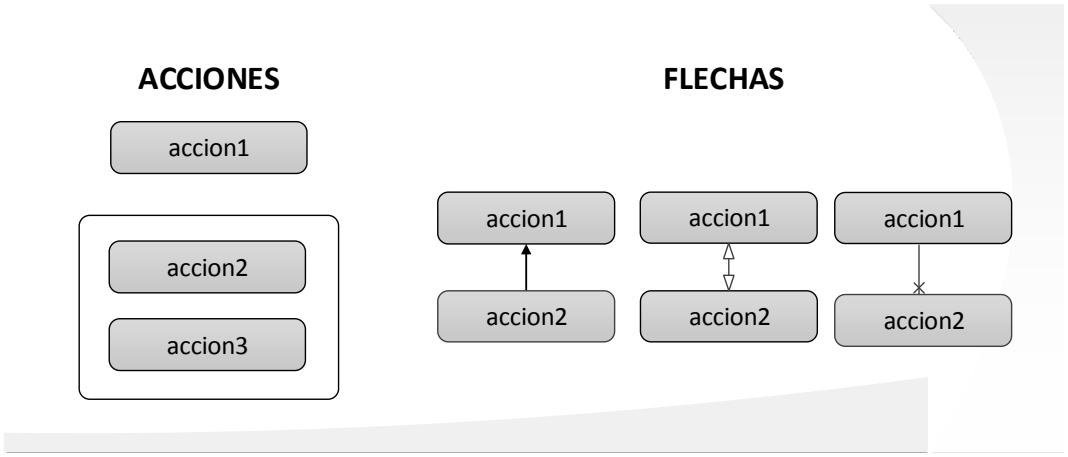

Figura 4.5: Tipos de componentes

4.1.4. Modelo del protocolo

En esta sección se describirá el proceso de la Transformación Bacteriana configurado en el tutor según la notación descrita en la sección anterior. Dada la naturaleza de la Transformación Bacteriana, es necesario colocar algún recipiente con una sustancia dentro del horno a 37°C consecutivas veces. Para omitir esa repetición, se ha creado el subproceso "Agitación en horno (X)" que se muestra en la figura 4.6 y que corresponde al proceso de agitación; como se puede apreciar, se tiene un objeto "X" como entrada y que viene a representar al recipiente.

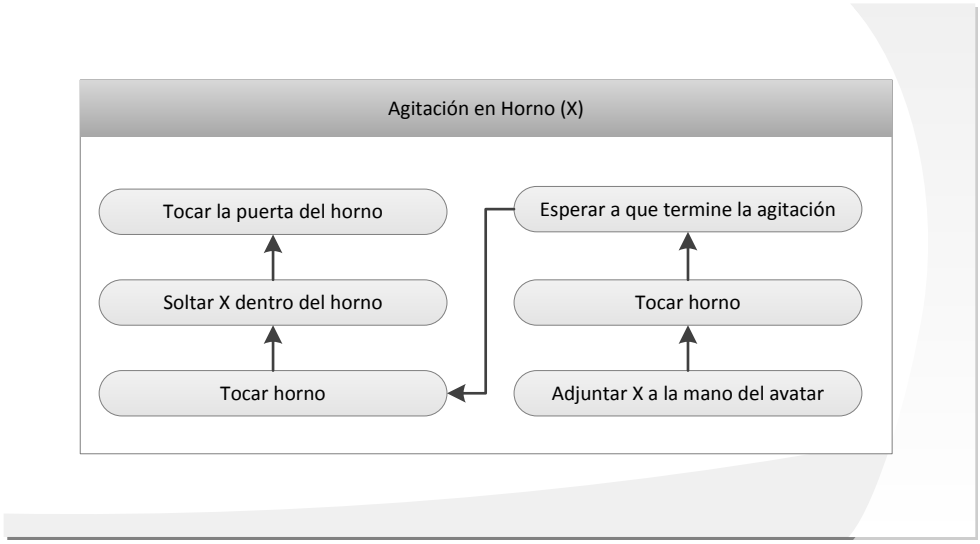

Figura 4.6: Subproceso de agitación en el horno.

Un subproceso similar se encuentra en el tomar y soltar un objeto dentro de la cabina, tales como la placa Petri, el tubo *Eppendorf*, la botella con medio de cultivo y el falcon;

sin embargo, para estos casos no se ha visto la necesidad de hacer un subproceso que incluya a todos debido a que presentan otras acciones intermedias.

La figura 4.7 corresponde al procedimiento completo de la Transformación Bacteriana, en el que se puede ver que el procedimiento puede comenzar con cualquier acción de las que están agrupadas en "a". Además, se puede observar que el subproceso "Agitación en horno (X)" ha sido utilizado para realizar la agitación del tubo *Eppendorf*, la placa Petri y el falcon. Para el caso de la bandeja de hielo, se desprende que si ocurre la acción "Soltar bandeja en cabina", la acción "Tomar bandeja" se anula puesto que el avatar ya no la tiene adjuntada a su mano, y viceversa, porque podría ocurrir que el usuario toma de nuevo la bandeja de la cabina para ir a buscar un nuevo químico.

Para agregar los químicos 1 y 2 a la cubeta de electroporación se necesita que previamente se hayan tomado; esto es debido a que la mezcla se inicia en la bandeja y sólo se pueden añadir a la mezcla aquellas sustancias que han sido tomadas previamente; como se puede observar en la figura 4.7, estas relaciones de dependencia también se ven reflejadas.

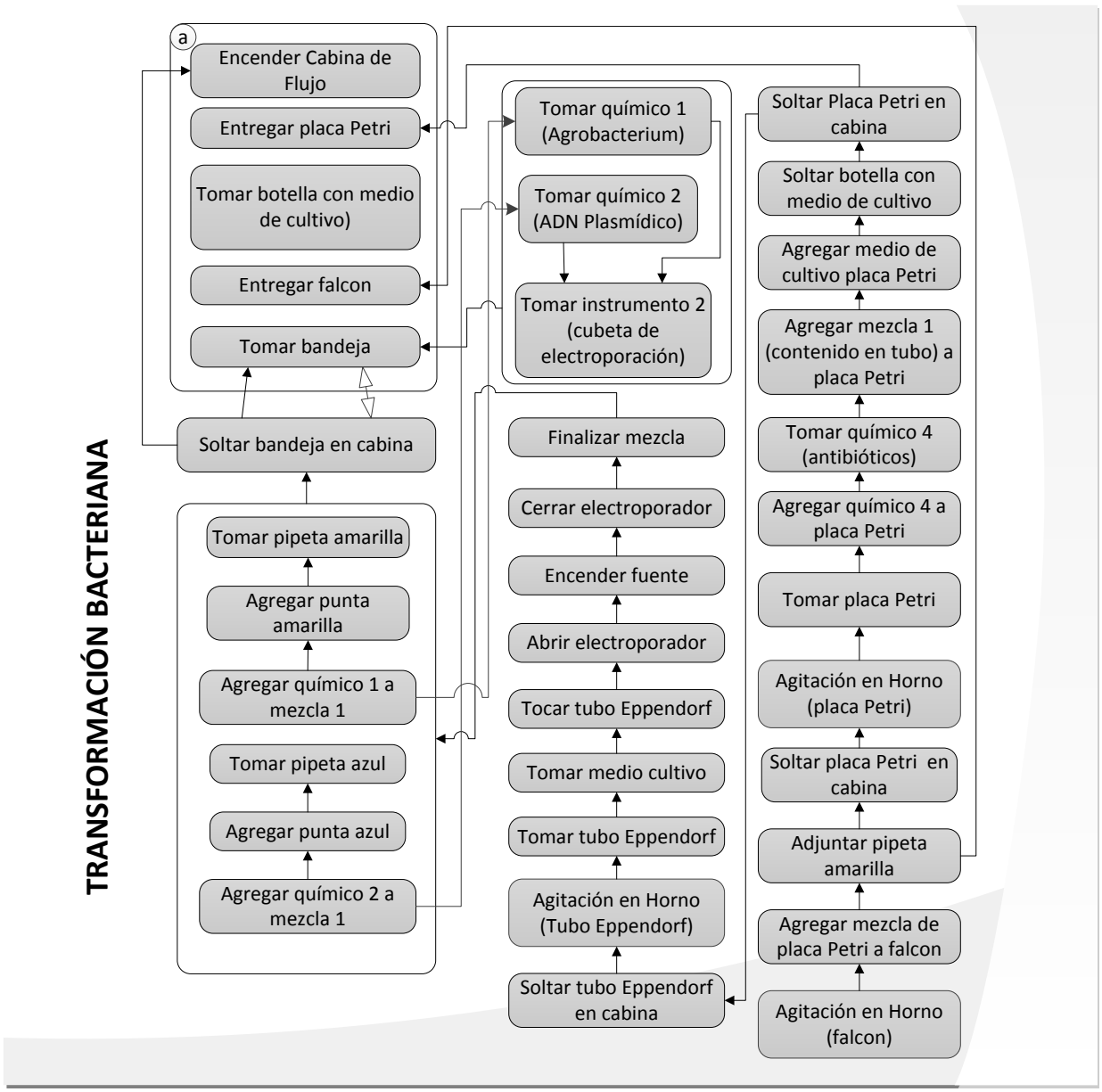

Figura 4.7: Protocolo de la Transformación Bacteriana.

4.1.5. Modelo de dominio

La figura 4.8 muestra la relación entre todos los objetos que están involucrados en la Transformación Bacteriana. Además, está representado el objeto tutor debido a que interactúa con todos los demás mediante las validaciones descritas en la sección 4.1.1.

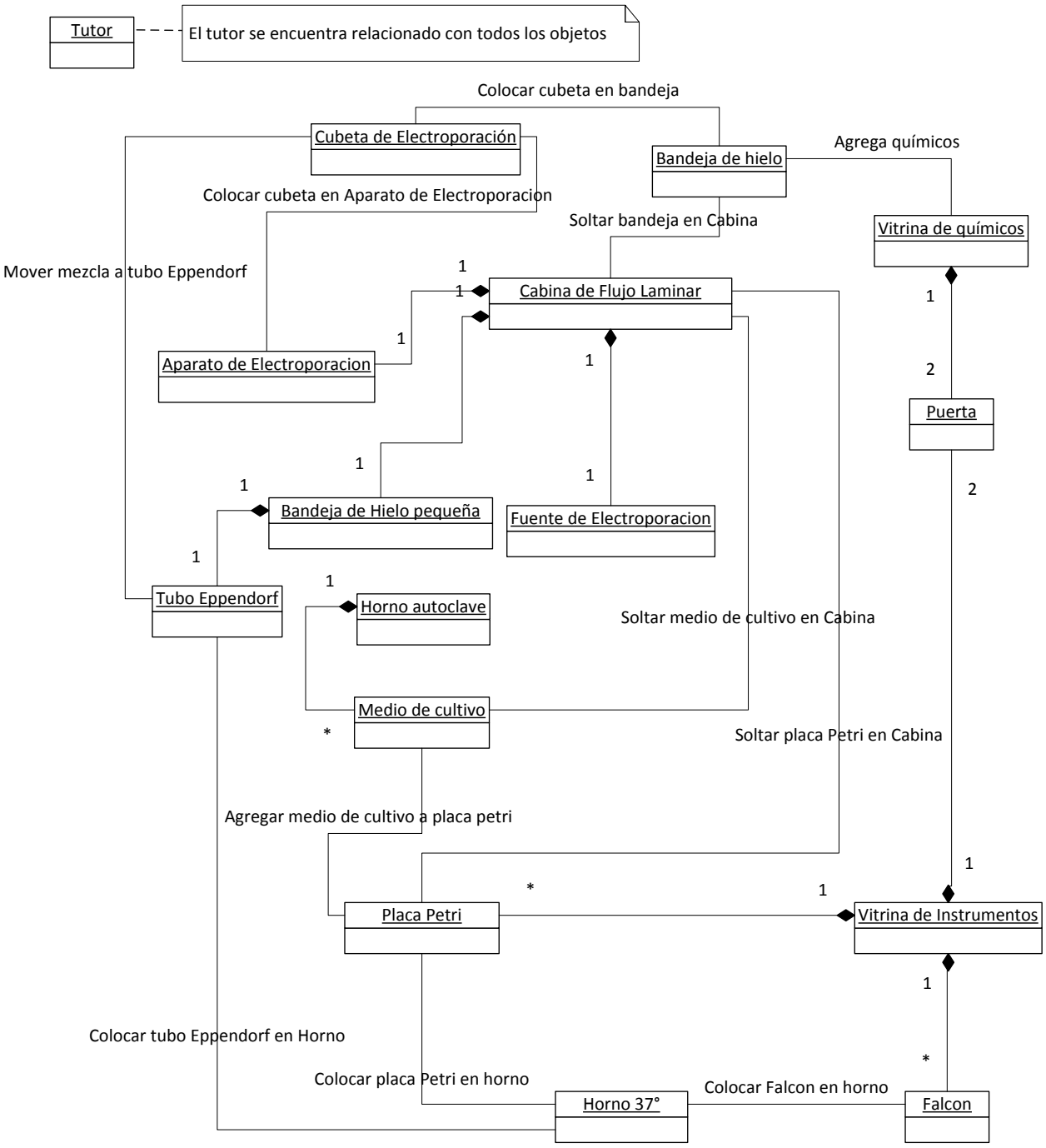

Figura 4.8: Modelo de dominio

4.1.6. Diseño de los objetos activos

Los objetos activos representan a aquellos objetos con comportamiento propio. Para el desarrollo de la Transformación Bacterianase ha necesitado la construcción o modificación de algunos objetos del mundo.

Las siguientes tablas están divididas en las siguientes cuatro columnas, tal como sigue:

- **Mensaje recibido:** Son los mensajes que el objeto escucha por un determinado canal, pueden recibir mensajes de otros objetos o los mensajes con validaciones del tutor.
- **Evento recibido:** Son los eventos que el avatar realiza sobre ellos, como soltar, tocar, adjuntar, etc.
- **Actividad disparada:** Es la acción que está configurada dentro del *script* para ser ejecutada cuando se recibe un cierto mensaje o evento.
- **Mensaje enviado:** Son los mensajes que envían las actividades disparadas.

Cabina de flujo laminar

Mensaje recibido	Evento recibido	Actividad disparada	Mensaje enviado
	tocar	validarEncenderCabina	validarEncenderCabina
encenderCabinaValido		encenderCabina validarTiempoFiltrarAire	bloquearEsterilizador validarContarTiempoCabina
comprobarCabinaAvatar	tocar	validarApagarCabina	validarApagarCabina
apagarCabinaValido		apagarCabina	

Cuadro 4.3: Descripción de la Cabina de Flujo

Dispensador de bandejas de hielo

Mensaje recibido	Evento recibido	Actividad disparada	Mensaje enviado
	tocar	validarEntregarBandeja	validarEntregarBandeja
entregarBandejaValido	adjuntarAvatar	entregarBandeja	

Cuadro 4.4: Descripción del Dispensador de bandejas de hielo

Bandeja de hielo

Mensaje recibido	Evento recibido	Actividad disparada	Mensaje enviado
	adjuntar	validarAdjuntarBandejaHielo	validarAdjuntarBandejaHielo
adjuntarBandejaHieloValido			
	soltar	borrarAdjuntarBandejaHielo validarSoltarBandejaPoyataPrimeraVez	validarSoltarBandejaPoyataPrimeraVez
soltarBandejaPoyataPrimeraVezValido		validarSoltarBandejaPoyata	validarSoltarBandejaPoyata
soltarBandejaPoyataValido			

Cuadro 4.5: Descripción de la Bandeja de hielo

Aparato de electroporación

Mensaje recibido	Evento recibido	Actividad disparada	Mensaje enviado
abrirPuerta		abrirTapa mostrarCubetaElectrop	
	tocarPuerta		validarCerrarElectrop
cerrarElectropValido	tocarPuerta	cerrarTapa validarAbrirEctrop	validarAbrirElectrop
abrirElectropValido		ocultarCubetaElectrop abrirTapa	

Cuadro 4.6: Descripción del aparato de electroporación

Tubo *Eppendorf*

Mensaje recibido	Evento recibido	Actividad disparada	Mensaje enviado
	tocar	validarAgregarMezclaTubo	validarAgregarMezclaTubo
agregarMezclaTuboValido		mostrarMezcla	
	tocar	validarEntregarTuboMezcla	validarEntregarTuboMezcla
adjuntarTuboValido	soltarTuboHorno	validarSoltarTubo	validarSoltarTubo
soltarTuboValido		cambiarPosicion	
entregarTubo		validarEntregarTubo	validarEntregarTubo
soltarTuboNoValido	adjuntarTuboAvatar		
entregarTuboValido	soltarTuboCab	validarSoltarTuboCabina	validarSoltarTuboCabina
soltarTuboCabinaValido		cambiarPosicion	
soltarTuboCabinaNoValido	adjuntarTuboAvatar		

Cuadro 4.7: Descripción de Tubo *Eppendorf*

Horno de autoclave

Mensaje recibido	Evento recibido	Actividad disparada	Mensaje enviado
	tocar	validarEntregarBotella	validarEntregarBotella
entregarBotellaValido		entregarBotella	

Cuadro 4.8: Descripción del Horno de autoclave

Placa Petri

Mensaje recibido	Evento recibido	Actividad disparada	Mensaje enviado
entregarPlacaPetri		entregarPlacaPetri	
	adjuntar	adjuntarPlacaPetri	
	soltarPlacaCabina	validarSoltarPlacaCabina	validarSoltarPlacaCabina
soltarPlacaCabinaValido	tocar	validarAgregarmediocultivo	validarAgregarmediocultivo
soltarPlacaCabinaNoValido	adjuntarPlacaAvatar		
agregarMedioCultivoValido	tocar	validarAgregarMezclaBact	validarAgregarMezclaBact
agregarMezclaBactValido	tocar	validarEntregarPlacaMezcla	validarEntregarPlacaMez
entregarPlacaMezValido	adjuntarPlacaAvatar soltarPlacaCabina	validarSoltarPlacaCab	validarSoltarPlacaCab
soltarPlacaCabValido			
soltarPlacaCabNoValido	adjuntarPlacaAvatar		

Cuadro 4.9: Descripción de la placa Petri

Falcon

Mensaje recibido	Evento recibido	Actividad disparada	Mensaje enviado
entregarFalcon		entregarFalcon	
	soltar	soltarFalconCabina validarSoltarFalconCabina	validarSoltarFalconCabina
validarSoltarFalconCabina NoValido	adjuntarFalconAvatar		
soltarFalconCabinaValido	tocar	validarAgregarMezclaFalcon	validarAgregarMezclaFalcon
agregarMezclaFalconValido	tocar	validarEntregarfalconMezcla	validarEntregarFalconMezcla
entregarFalconMezclaValido	soltarEnHorno	validarSoltarFalconHorno	validarSoltarFalconHorno
soltarFalconHornoNoValido	adjuntarFalconAvatar		
soltarFalconHornoValido			
entregarFalconHorno	adjuntarFalconAvatar		

Cuadro 4.10: Descripción de Falcon

Horno a 37°

Mensaje recibido	Evento recibido	Actividad disparada	Mensaje enviado
	tocar	validarAbrirPuerta	validarAbrirPuerta
abrirPuertaValido	tocar	bloquear validarCerrarPuerta	validarCerrarPuerta
cerrarPuertaValido	tocar	validarEncender	validarEncender
encenderValido	tocar	validarAbrirPuertaPetri	validarAbrirPuertaPetri
abrirPuertaPetriValido	tocar	validarCerrarPuertaPetri	validarCerrarPuertaPetri
cerrarPuertaPetriValido	tocar	validarEncenderPetri	validarEncenderPetri
encenderPetriValido	tocar	validarAbrirPuertaFalcon	validarAbrirPuertaFalcon
abrirPuertaFalconValido	tocar	validarCerrarPuertaFalcon	validarCerrarPuertaFalcon
cerrarPuertaFalconValido	tocar	validarEncenderFalcon	validarEncenderFalcon
encenderFalconValido			

Cuadro 4.11: Descripción del horno a 37°C

Vitrina de instrumentos

Mensaje recibido	Evento recibido	Actividad disparada	Mensaje enviado
	tocar	abrirPuerta, crearDialogoInstrumental	
	seleccionarFalcon	validarSeleccionarFalcon	validarSeleccionarFalcon
entregarFalconValido		cerrarPuerta	
	tocar	abrirPuerta, crearDialogoInstrumental	
	seleccionarCubeta	validarSeleccionarCubeta	validarSeleccionarCubeta
entregarCubetaValido		cerrarPuerta	
	tocar	abrirPuerta, crearDialogoInstrumental	
	seleccionarFalcon	validarSeleccionarFalcon	validarSeleccionarFalcon
entregarFalconValido		cerrarPuerta	
	tocar	abrirPuerta, crearDialogoInstrumental	
	seleccionarAntibiotico	validarSeleccionarAntibiotico	validarSeleccionarAntibiotico
entregarAntibioticoValido		cerrarPuerta	

Cuadro 4.12: Descripción de la Vitrina de instrumentos

Frigorífico

Mensaje recibido	Evento recibido	Actividad disparada	Mensaje enviado
	tocar	abrirPuerta, crearDialogoFrigorifico	
	seleccionarADNPlasm	validarEntregarADNPlasm	validarEntregarADNPlasm
entregarADNPlasmValido		cerrarPuerta	
	tocar	abrirPuerta, crearDialogoFrigorifico	
	seleccionarAgrobac	validarEntregarAgrobac	validarEntregarAgrobac
entregarAgrobacValido		cerrarPuerta	
	tocar	abrirPuerta, crearDialogoFrigorifico	
	seleccionarMedioCult	validarEntregarMedioCult	validarEntregarMedioCult
entregarMedioCultValido		cerrarPuerta	

Cuadro 4.13: Descripción del frigorífico

Base pipeta amarilla

Mensaje recibido	Evento recibido	Actividad disparada	Mensaje enviado
	tocar	validarEntregarPipetaAmarilla	validarEntregarPipetaAmarilla
entregarPipetaAmarillaValido	adjuntarAvatar	entregarPipetaAmarilla	

Cuadro 4.14: Descripción de la pipeta amarilla

Caja de puntas amarillas

Mensaje recibido	Evento recibido	Actividad disparada	Mensaje enviado
	tocar	validarCargarPuntaAmarilla	validarCargarPuntaAmarilla
entregarPipetaAmarillaValido		cargarPuntaAmarilla	

Cuadro 4.15: Descripción de la caja de puntas amarillas

Fuente de voltaje

Mensaje recibido	Evento recibido	Actividad disparada	Mensaje enviado
	tocar	validarEncenderFuenteElectrop	validarEncenderFuenteElectrop
encenderFuenteElectropValido			

Cuadro 4.16: Descripción de la fuente de voltaje

4.1.7. Diseño de la interacción entre objetos

En esta sección se representarán todas las interacciones del avatar con los objetos mediante diagramas de secuencia. Como se puede ver en cada uno de ellos, el avatar y el tutor siempre están presentes debido a que el primero inicia una cadena de acciones, mientras que el segundo, realiza todas las validaciones requeridas.

En algunos diagramas existen procesos alternativos para soltar un objeto, en caso no se cumplan ciertas validaciones el objeto debe ser adjuntado nuevamente al avatar, caso contrario, se coloca en la base correspondiente.

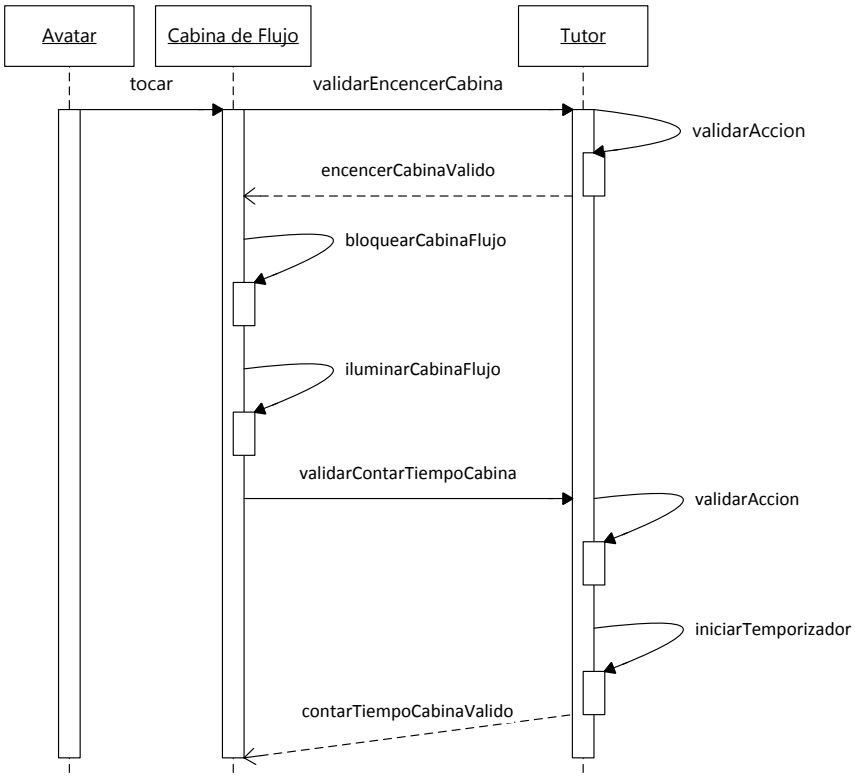

Figura 4.9: Diagrama de secuencia - Encender Cabina de Flujo

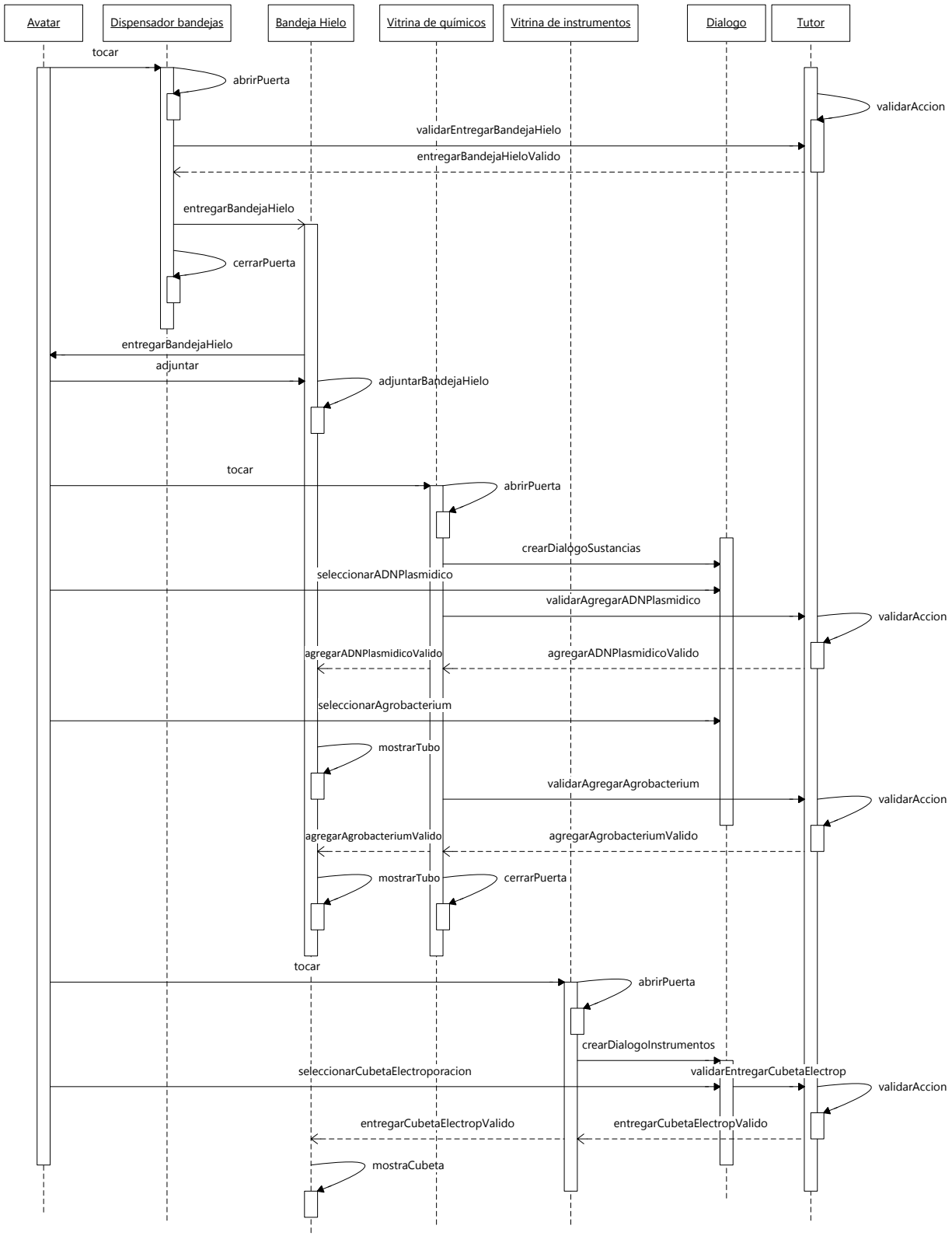

Figura 4.10: Diagrama de secuencia - Agregar elementos a bandeja de hielo

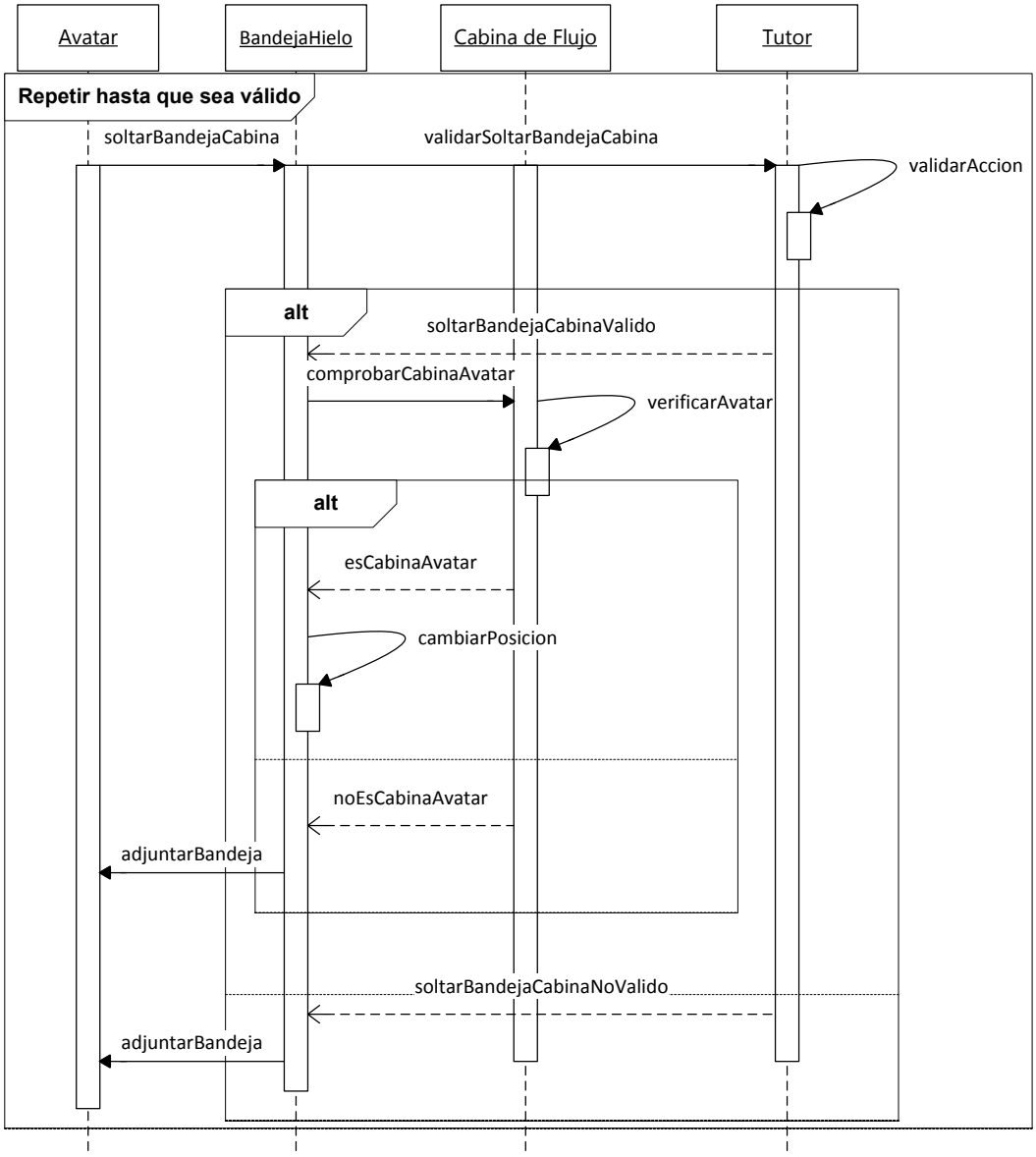

Figura 4.11: Diagrama de secuencia - Soltar bandeja de hielo en Cabina

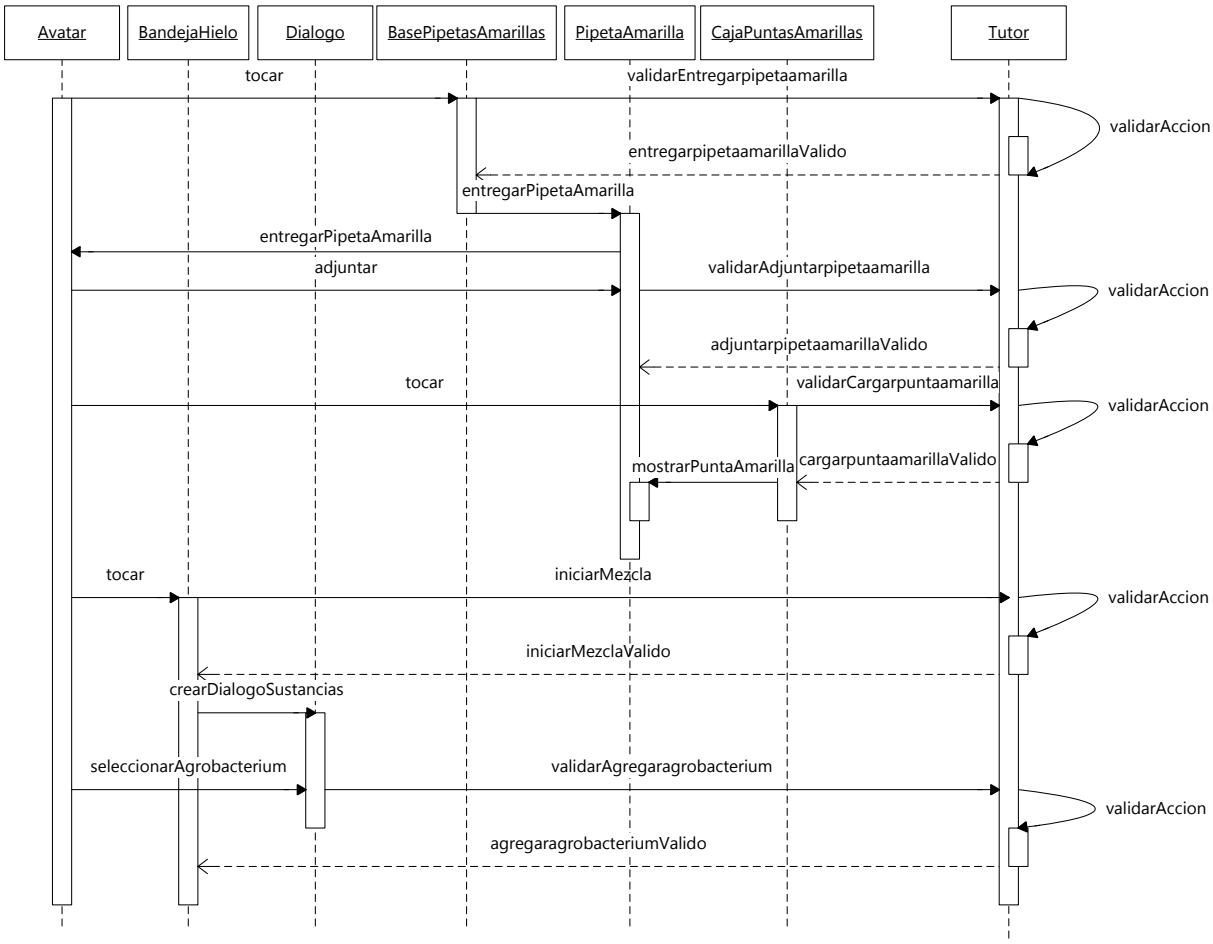

Figura 4.12: Diagrama de secuencia - Agregar Agrobacterium a Cubeta

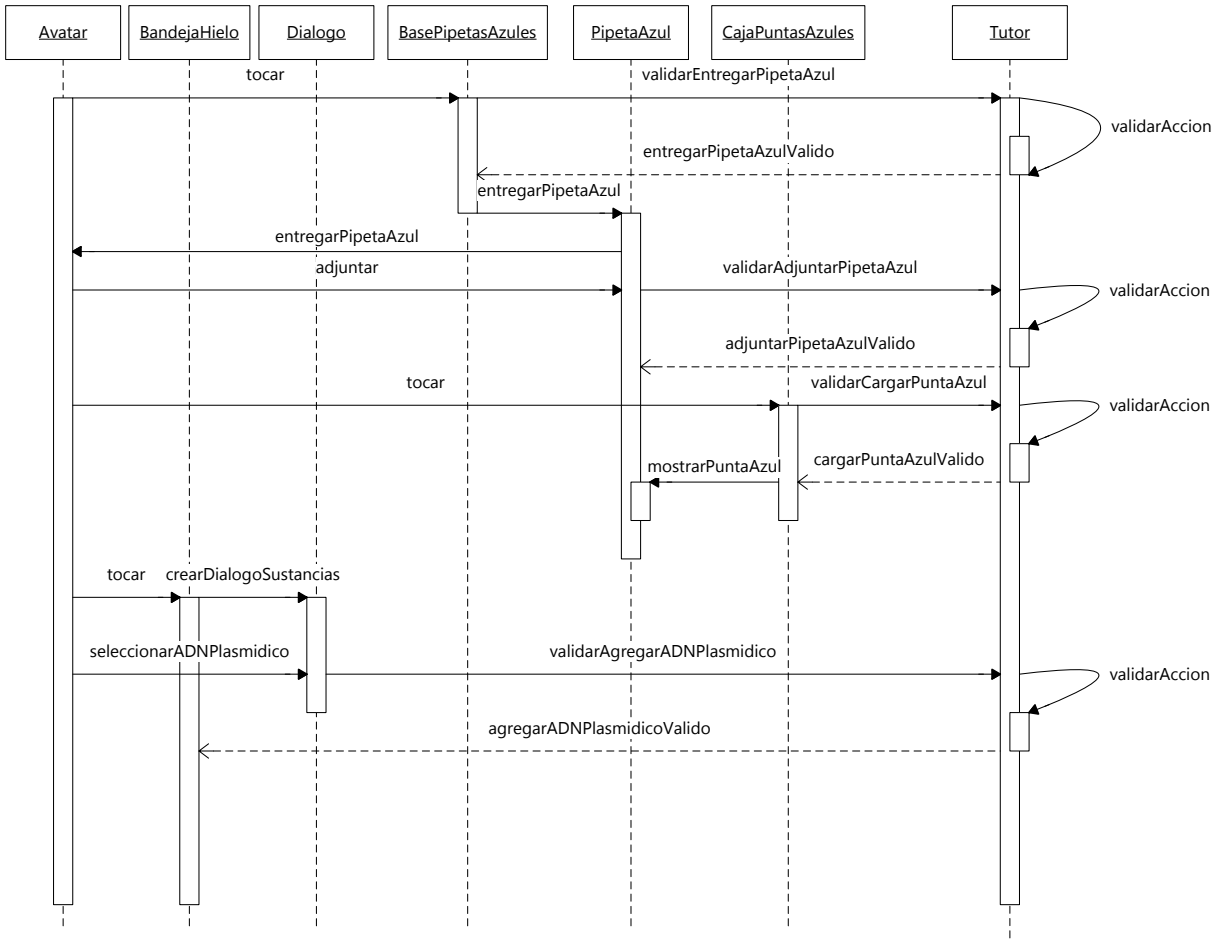

Figura 4.13: Diagrama de secuencia - Agregar ADN Plasmídico a Cubeta

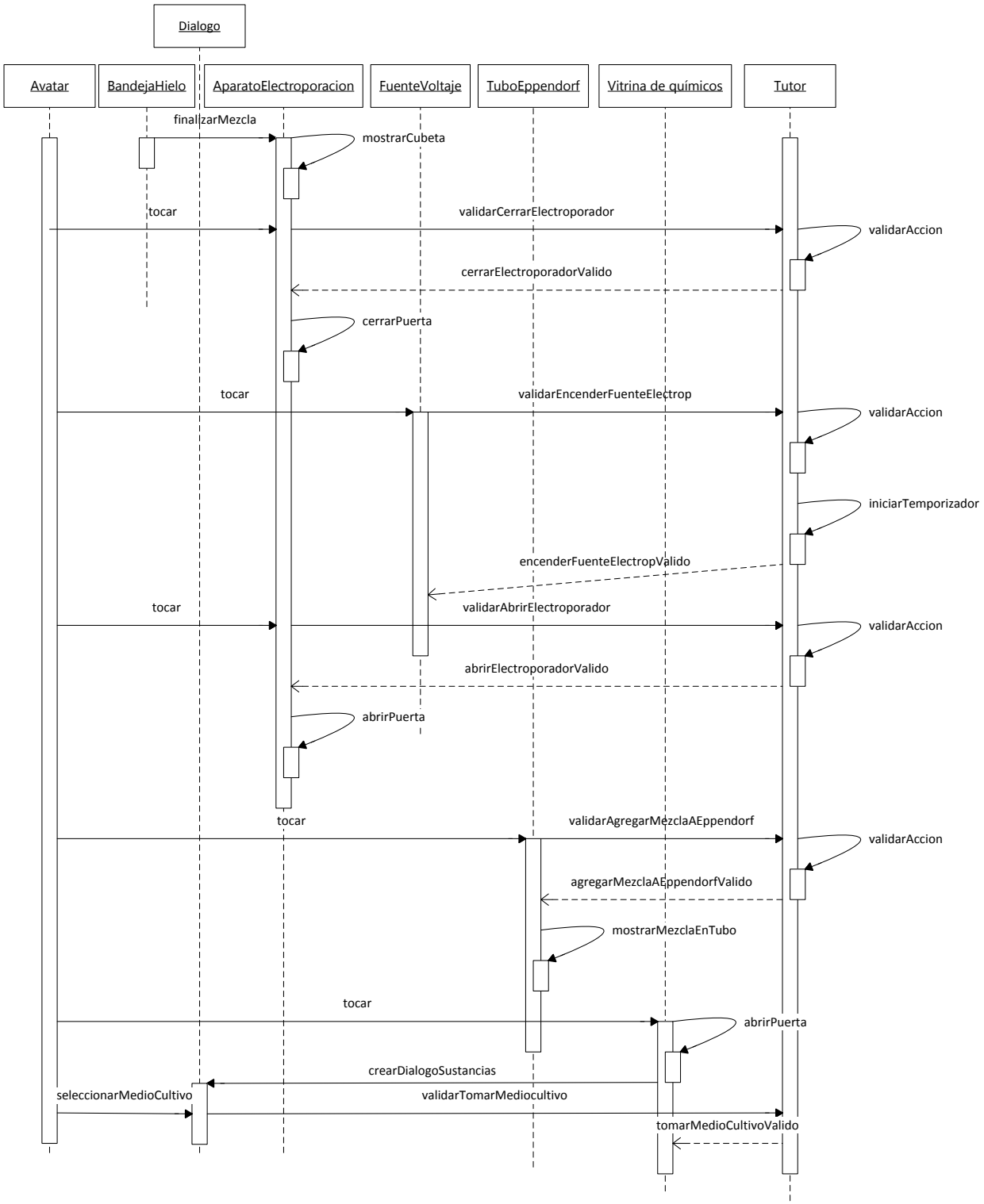

Figura 4.14: Diagrama de secuencia - Electroporación

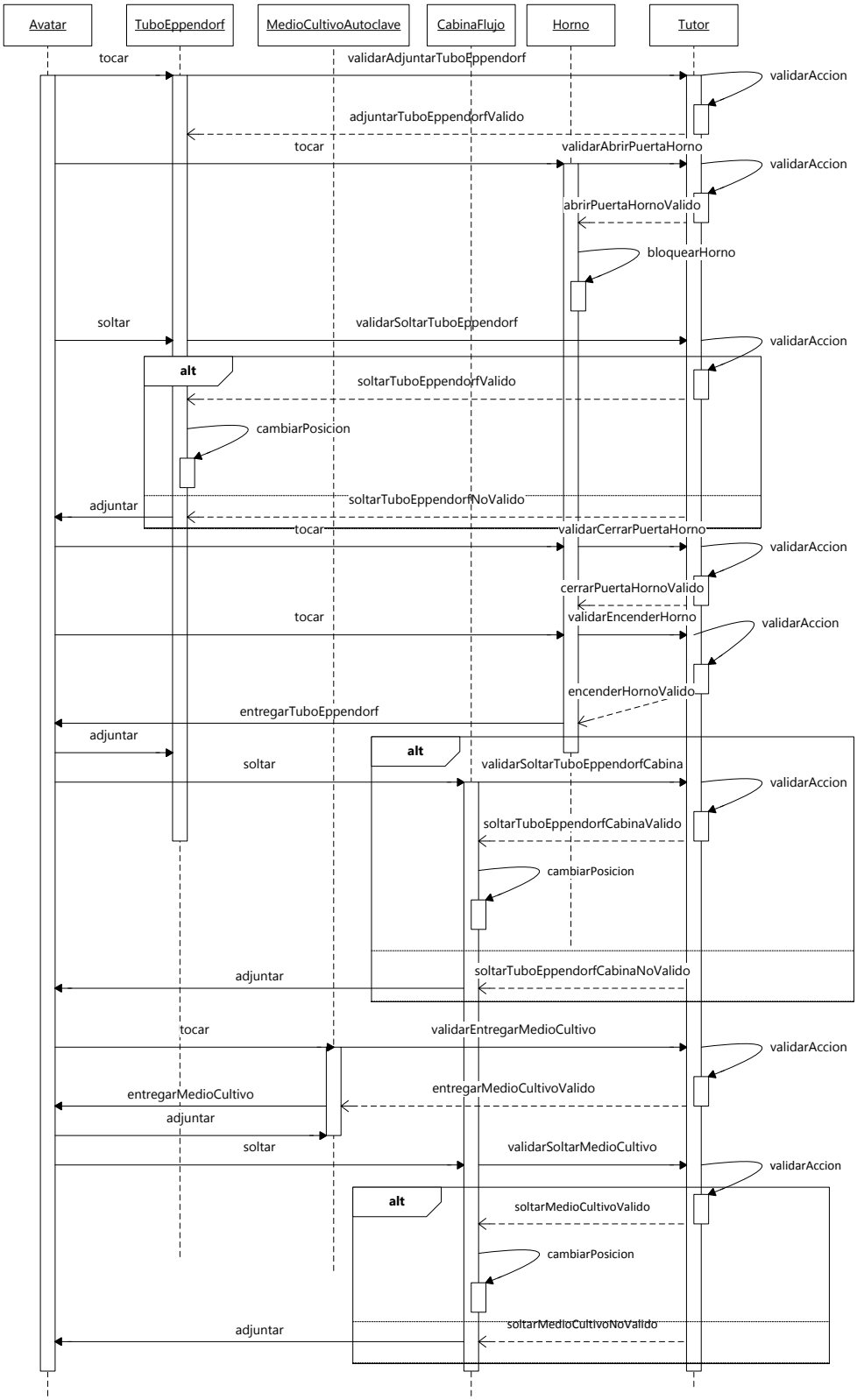

Figura 4.15: Diagrama de secuencia - Agitación Tubo *Eppendorf*

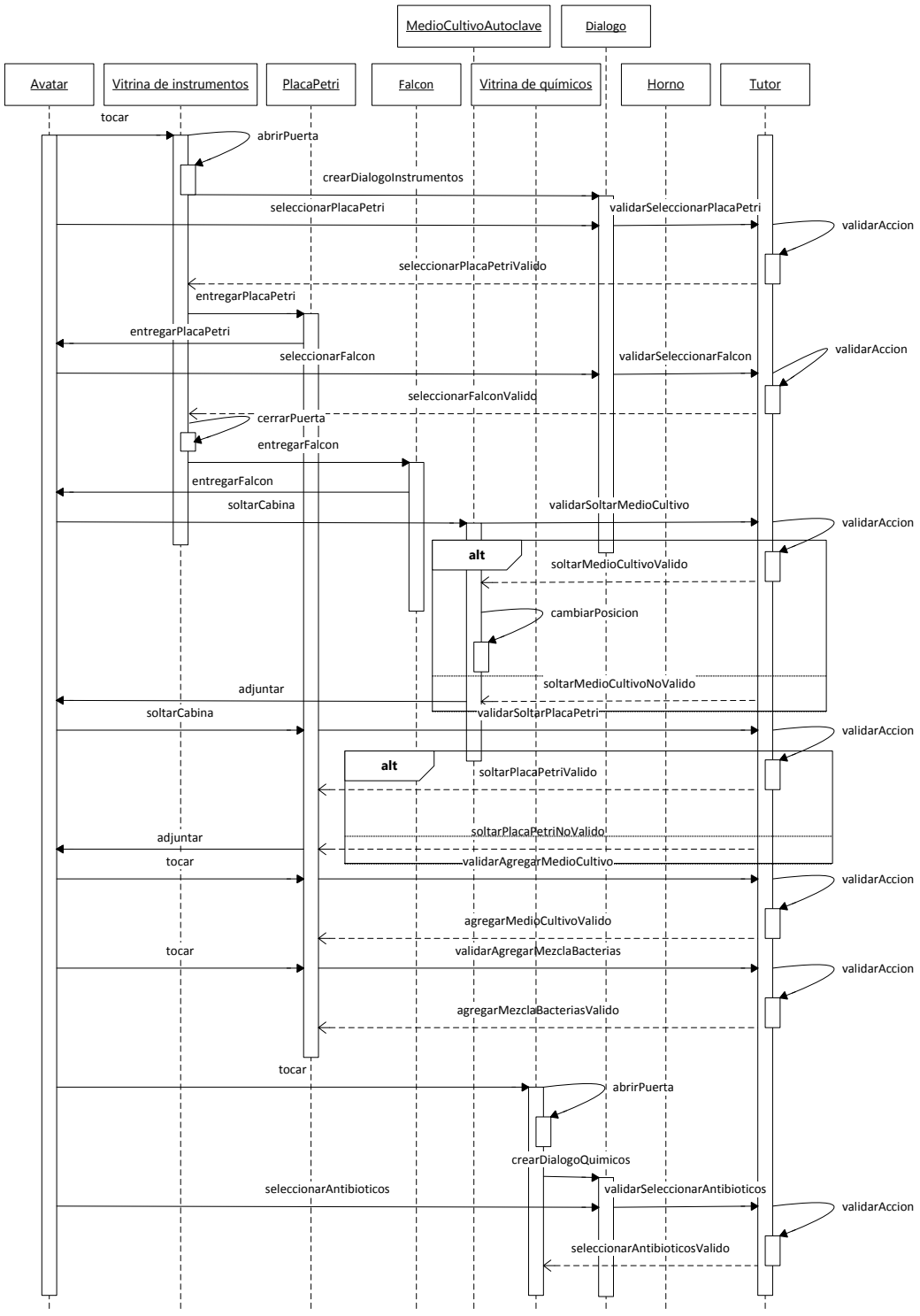

Figura 4.16: Diagrama de secuencia - Mezcla en Placa Petri

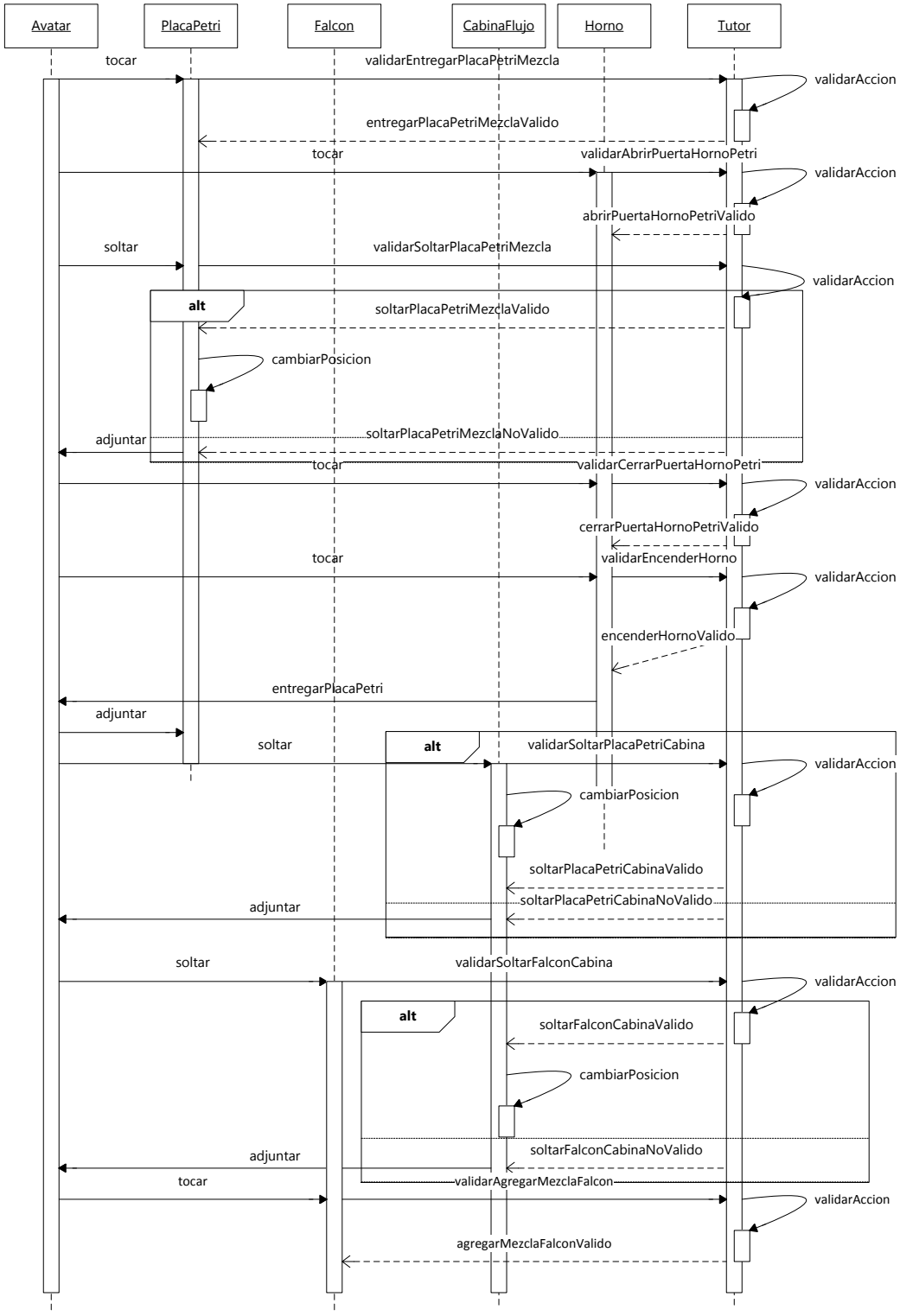

Figura 4.17: Diagrama de secuencia - Agitación Placa Petri

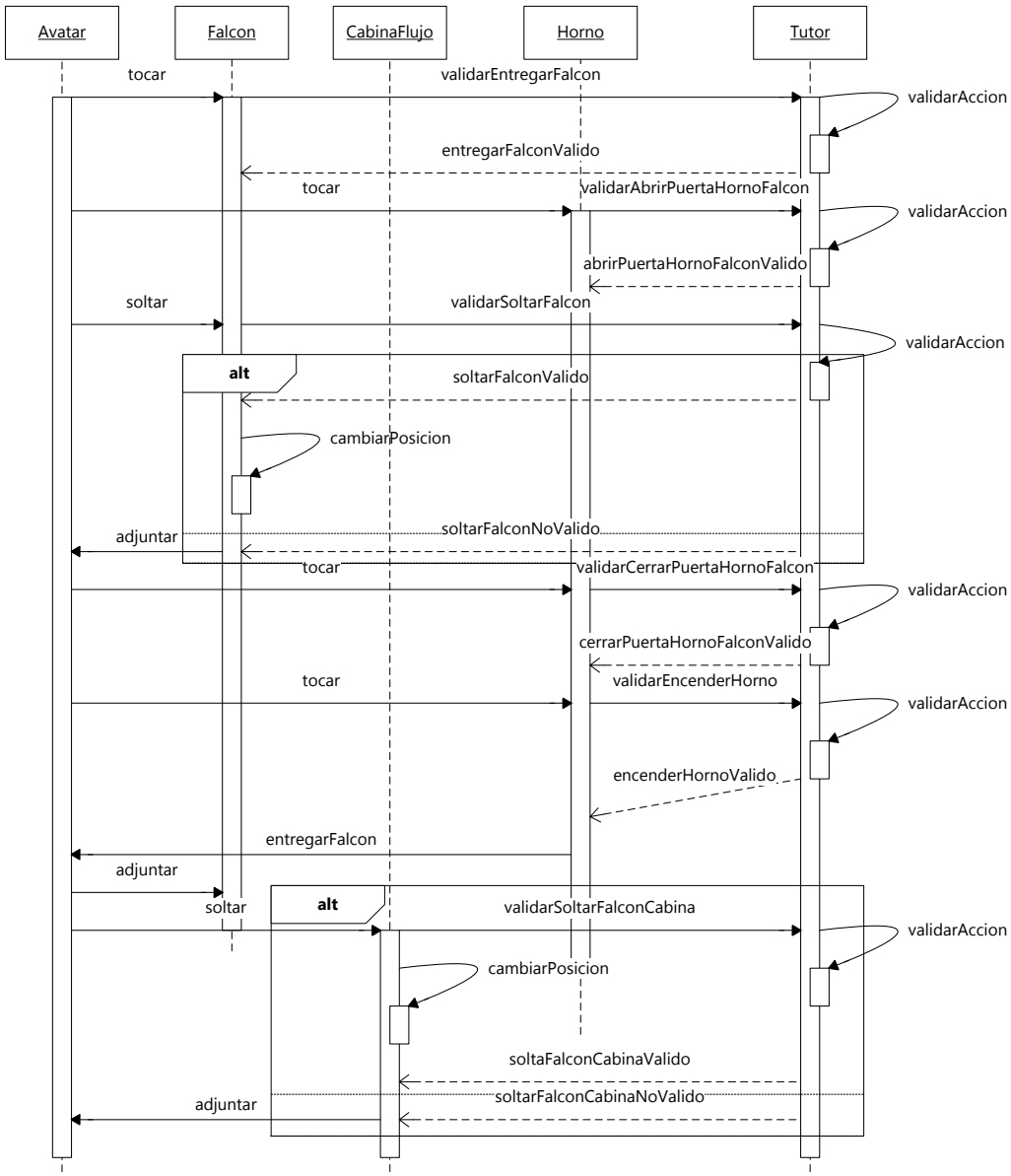

Figura 4.18: Diagrama de secuencia - Agitación Falcon

4.2. Cambios realizados en el visor Firestorm

Definidos los aspectos que ofrecen cierta problemática a la usabilidad, en esta sección se describirán los cambios que se hicieron en el visor con el fin de mitigarlos. Como se mencionó anteriormente, Firestorm posee la ventaja de poder modificar su código fuente ya que se puede descargar desde su web oficial y, además, presenta las siguientes características:

- Está desarrollado en lenguaje *C++*.

- Utilización de *Microsoft Visual Studio 2010* como entorno de desarrollo.
- Su estructura está dividida en varios proyectos, lo que promueve la alta cohesión y el bajo acoplamiento.
- Utilización de *Mercurial* como herramienta de administración de código fuente.
- Utilización de *CMake* como herramienta de generación de código fuente.
- Utilización de *Python* para la ejecución de algunas tareas en la descarga del código fuente.

Las herramientas de software mencionadas son necesarias para la descarga y generación de la solución[1] de Firestorm (el proceso detallado se encuentra en el Anexo A.1). En la práctica, cuando se inició la descarga de los archivos, se tuvieron muchas complicaciones debido a las configuraciones o las versiones de las herramientas, además, surgieron inconvenientes al momento de generar la solución a través del CMake y Python, debido a que se requerían librerías, *plug-ins* adicionales o algunas modificaciones en el código, que no estaban documentadas en ningún sitio.

Una vez que se tuvo la solución completa y tras verificar que compilaba correctamente y sin errores, se procedió a revisar su estructura para entender la distribución de los proyectos. En esta revisión se observó que la solución contaba con:

- Un instalador que se creaba luego de completar la compilación y que se creaba en la carpeta "bin" de la configuración con la que se compilaba (DebugOS, RelWithDebInfoOS o ReleaseOS).
- Archivos XML con el listado de los servidores, temas y otras configuraciones en el proyecto *firestorm-bin*. Algunos de estos archivos se modificaban a través de la versión instalada de Firestorm, desde la opción de "Configuración"; mientras que otros, representaban las opciones de menú, las etiquetas o cualquier texto que se mostraba en el visor.
- La versión inicial no contaba con soporte para OpenSim, tan solo para SecondLife. Esto fue resuelto luego de que se modificara una línea en el archivo CMakeLists.txt, específicamente: HAS_OPENSIM_SUPPORT "OpenSim support" ON. Previamente se encontraba en "OFF", se hizo el cambio a "ON" y se volvió a ejecutar el proceso de generación de archivos.

[1] Agrupador de distintos proyectos con los que se trabajan en *Microsoft Visual Studio 2010*.

- Tres tipos de configuraciones, DebugOS, RelWithDebInfoOS y ReleaseOS. La versión Debug y RelWithDebInfoOS generaban una consola con la información de la depuración, tanto al momento de hacer debug, como en la versión instalada. La versión Debug generaba archivos con información del estado de la depuración y el proyeco (archivos pdb). Por otra parte, la versión ReleaseOS sólo incluía el visor.

Posteriormente a la identificación de los puntos mencionados, se empezó con el trabajo de adecuar el código para cubrir los problemas de usabilidad que se habían detectado. Las carpetas que fueron relevantes para las modificaciones de código se encuentran en el Apéndice A.2. Las modificaciones hechas sobre los archivos fuente se realizaron aplicando las siguientes estrategias:

1 **Depuración de código**
La ventaja de utilizar *Microsoft Visual Studio 2010* fue que su entorno permitía una depuración mientras la aplicación estaba siendo ejecutada; de esta manera, se podía identificar con mayor facilidad la línea de código que se tenía que modificar. Esta estrategia sirvió para modificar archivos con código C++ (Extensión *.cpp* o *.h*).

2 **Búsqueda por Clave/Valor**
Debido a la revisión previa, se sabía que algunas personalizaciones podían ser hechas desde la ventana de Configuración de Firestorm, así que, para que, pudieran tener otros valores predeterminados se cambió el idioma del visor a inglés y, posteriormente, se empezó la búsqueda por medio de los textos de las etiquetas. En primera instancia se buscaron estas claves en los ficheros XML y en segunda, en todos los archivos de la solución. Cabe mencionar que para el caso de los archivos XML de los temas/skins, en la estructura de la solución sólo se mostraban las configuraciones para el idioma inglés; una vez identificado el archivo, se procedió a la búsqueda de su homólogo en la carpeta del idioma español (*XUI_ES_PATH*).

Detallados todos los puntos anteriores, se procederá a explicar todas las adecuaciones en el visor de acuerdo a los puntos señalados en la Sección 3.4 (Problemas de Usabilidad con el visor Firestorm) del Capítulo 3 (Planteamiento del Problema).

4.2.1. Bloqueo a la práctica

Como ya se mencionó, en este punto el problema radicaba en que si el usuario presionaba el botón “Descartar” o el botón “Bloquear al propietario” de la ventana de confirmación que se mostraba cuando se entregaba un objeto al inventario, la práctica se volvía imposible de completar ya que las acciones posteriores requerían tener el objeto adjuntado

a la mano. Para evitar este problema, se decidió eliminar el mensaje de confirmación y simular que el usuario presionaba el botón "Permitir".

Aplicando la estrategia de **Depuración de código**, se logró identificar el punto que se debía modificar. Luego de su análisis se detectó lo siguiente:

- La clase ***LLOfferInfo*** representa a los objetos que son entregados al inventario del avatar.
- La propiedad ***mFromID***, de la instancia ***info***, representa el ID del objeto que está siendo entregado.
- La variable ***gAgentID*** representa el ID del avatar.
- La constante ***IOR_ACCEPT*** indica que el objeto está siendo aceptado para ser entregado al inventario.

Identificado lo anterior, se procedió a la adición del código necesario para que no se muestre el mensaje de confirmación. Se modificó el método ***inventory_offer_handler*** de la clase ***LLPostponedOfferNotification*** tal como se muestra a continuación:

```
class LLPostponedOfferNotification: public LLPostponedNotification
{
    ...
    void inventory_offer_handler(LLOfferInfo* info)
    {
        //=============================================================================
        //Código preexistente.
        if (LLMuteList::getInstance()->isMuted(info->mFromID, info->mFromName) || ↩
            LLMuteList::getInstance()->isMuted(LLUUID::null, info->mFromName))
        {
            info->forceResponse(IOR_MUTE);
            return;
        }
        //=============================================================================

        //=============================================================================
        //Se agregaron las siguientes líneas para no mostrar el mensaje de confirmación
        //al momento de añadir un ítem al inventario.
        if((info->mFromID != gAgentID))
        {
            info->forceResponse(IOR_ACCEPT);
            return;
        }
        //=============================================================================
        ...
```

Luego de este cambio se hicieron varias pruebas en distintos casos para verificar que el objeto entregado al inventario no requería el mensaje de confirmación. Finalmente se dió por superado este problema que podría ser considerado como el más crítico de todos.

4.2.2. Mensajes innecesarios

En este punto se deseaban eliminar aquellos mensajes que no ofrecían información importante para el usuario, sino, más bien, confundirle. Dentro de los mensajes existentes se encontraban:

- El mensaje que se mostraba luego de agregar un objeto al inventario.
- El mensaje que se mostraba cuando el puntero se posicionaba sobre un objeto. Se detallaban los números de prims, la posicion y la distancia al avatar.

El primer punto fue resuelto en la adecuación del punto anterior (Bloqueo a la práctica), mientras que en el segundo sí había que aplicar las estrategias de **Depuración de código** y **Búsqueda por Clave/Valor**; la segunda, para identificar la clave en el código C++, y la primera, para verificar que el mensaje sí que efectivamente se construía en esa sección del código.

Luego de sucesivas pruebas, se localizó el código que se tenía que modificar, se agregó una instrucción de tipo ***if*** con la condición en ***false*** para que no se construyera el mensaje y, tampoco, se mostrara. La modificación se realizó en el método ***handleTooltipObject*** de la clase ***LLToolPie*** tal como se muestra a continuación:

```
BOOL LLToolPie::handleTooltipObject( LLViewerObject* hover_object, std::string line↩
    , std::string tooltip_msg)
{
    ...
    //Aproximadamente la línea 1235.
    //========================================================================================
    if(false) //Se agregó esta línea para que no construyera el mensaje.
    {
        //En esta sección del código se obtenían los valores de las propiedades
        //que se iban a mostrar.

        // Get prim count
        S32 prim_count = LLSelectMgr::getInstance()->getHoverObjects()->↩
            getObjectCount();
        ...
    }
    //========================================================================================
    ...
}
```

4.2.3. Soltar objeto adjuntado con mayor facilidad

Otro problema crítico que fue explicado en la Sección 3.4.3 (Soltar objeto adjuntado con mayor facilidad) se originaba en la naturaleza de la acción de la práctica. Como ya

se mencionó, algunas acciones requerían soltar un objeto adjunto en la mano del avatar, en determinado lugar o dentro de otro objeto contenedor. Para tratar de solucionar este problema se sugirió la posibilidad de imitar el concepto del botón "Levantarse" o "Dejar de volar"; ambos botones se ubicaban en la parte inferior izquierda y sólo se mostraban cuando el avatar se sentaba o volaba, respectivamente. Aplicando la estrategia de **Depuración de código**se logró determinar cómo funcionaba el botón "Dejar de volar"; en resumen, la funcionalidad estaba construida de la siguiente manera:

1. El panel que contenía el botón "Dejar de volar"era una instancia estática de la clase ***LLPanelStandStopFlying***, la que se heredaba de la superclase ***LLPanel***.

2. El botón "Dejar de volar"poseía el método ***setVisible*** que lo mostraba u ocultaba debido a un único parámetro de tipo ***boolean***.

3. La instancia de la clase ***LLPanelStandStopFlying*** hacía una llamada al método ***setStandStopFlyingMode*** cuando el avatar empezaba o dejaba de volar, según sea el caso.

4. En el método del punto anterior se hacía una llamada al método ***setVisible*** para mostrarlo u ocultarlo.

5. Finalmente, existía el archivo XML ***panel_stand_stop_flying.xml*** que describía el botón "Dejar de volar".

Entendida la construcción y funcionalidad se trató de averiguar si existía algún punto común en la construcción y visualización del botón. Dado que la acción de volar o de sentarse eran provocadas por clases muy especializadas y a las cuales, no se les podía modificar directamente para obtener la adecuación deseada, se procedió a reproducir su comportamiento, adaptándolo a la necesidad de soltar un objeto.

Para imitar la funcionalidad se decidió crear la clase ***LLWearObjectPanel***, cuyo diagrama de clases se muestra en la figura 4.19.

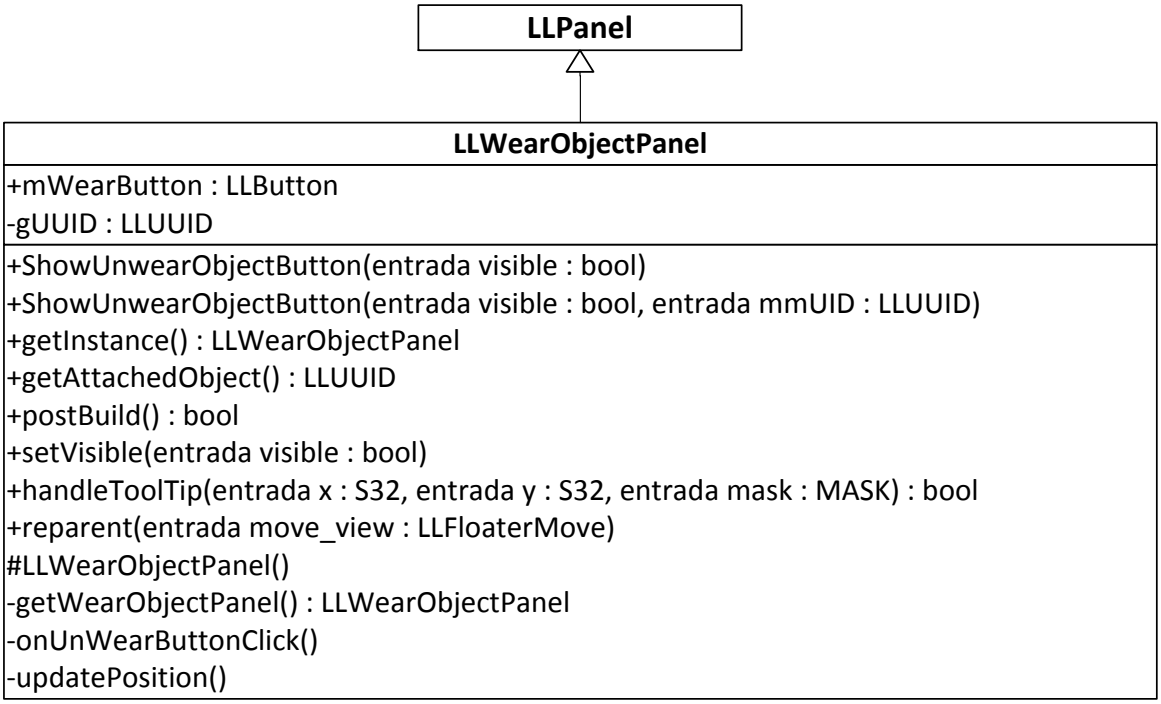

Figura 4.19: Diagrama de clases de ***LLWearObjectPanel***.

Como todas las instancias de las clases de Firestorm, se usa el patrón ***Singleton***[2]. Dado que la clase heredaba de ***LLPanel***, algunos métodos se sobreescribían y otros, eran completamente nuevos, tal como se describe en el Apéndice A.3.

Posteriormente, se creó el archivo XML ***panel_wear_object.xml*** con la siguiente información:

```
<panel name="panel_wear_object">
    <button label="Soltar" name="unwear_btn" tool_tip="Pulsa aqui para soltar el objeto↩
        ."/>
</panel>
```

La codificación completa de esta clase se encuentra en el Apéndice B.1; sin embargo, se detallarán algunas líneas importantes que se encuentran en el método ***callback_attachment_drop_custom***.

[2]Patrón de diseño que limita la creación de objetos a una sola vez. El valor se almacena en una variable estática y es la que se devuelve cuando se llama al método ***getInstance*** en Firestorm.

```
void callback_attachment_drop_custom(const LLSD& notification, const LLSD& response)
{
    ...

    //=============================================================================
    //En estas líneas se obtiene el identificador del objeto que
    //el avatar tiene adjuntado a su mano.
    LLUUID object_id;
    object_id = LLWearObjectPanel::getAttachedObject();
    LLInventoryItem* item = gInventory.getItem(object_id);
    LLViewerObject *object = gAgentAvatarp->getWornAttachment(item->getUUID());
    //=============================================================================

    ...//Validaciones

    //=============================================================================
    //En estas líneas se procede a soltar el objeto y
    //a ocultar el botón ''Soltar''
    LLSelectMgr::getInstance()->selectObjectAndFamily(object);
    LLSelectMgr::getInstance()->sendDropAttachment();
    LLWearObjectPanel::getInstance()->mWearButton->setVisible(FALSE);
    //=============================================================================

    return;
}
```

En la figura 4.20 se muestra cómo quedó el botón soltar luego de la adecuación. Como se puede ver, el usuario ya no tiene la necesidad de girar la cámara para poder darle clic al objeto adjuntado.

Figura 4.20: Vista del botón soltar.

Validaciones adicionales

El comportamiento básico para el botón "Soltar" ya estaba programado; sin embargo, al momento de las pruebas, se detectaron los siguientes casos en los que se necesitaban más adecuaciones.

- Debido a validaciones en algunas acciones, se forzaba el adjuntado automático del objeto nuevamente. En este caso el botón "Soltar" ya no se volvía a mostrar, a pesar de que el avatar tenía adjuntado en la mano el objeto
- Cuando el usuario soltaba el objeto a través de su menú contextual, el botón "Soltar" no desaparecía.

Las modificaciones al código que solucionan los problemas mencionados, se explican con más detalle en el Apéndice B.2. De igual manera, se aplicó la estrategia de **Depuración de código** para encontrar las líneas a ser cambiadas.

4.2.4. Configuraciones por defecto

En esta parte de la solución se modificarán los valores por defecto para algunas configuraciones. Esta sección se ha dividido en Conexiones y Temas. Cada uno de ellos contiene adecuaciones en archivos XML y en algunas líneas de código; estas modificaciones fueron hechas aplicando la estrategia **Búsqueda por Clave/Valor**.

Conexiones

Como se había explicado anteriormente, Firestorm cuenta con conexiones a servidores de tipo Second Life u Open Sim, cada uno de ellos cuenta con grids por defecto, por lo que se debía de agregar manualmente el servidor del laboratorio de Biotecnología. Para poder cubrir este requisito, se procedió a lo siguiente:

- Se modificó la constante ***MAINGRID*** en los archivos de cabecera: *NEWVIEW_PATH\fsgridhandler.h* y *NEWVIEW_PATH\llviewernetwork.h* para que "GridLab UPM" sea el grid por defecto.

```
#define MAINGRID "gridlab upm"
```

- Se modificó el archivo *NEWVIEW_PATH\app_settings\grids.fallback.xml* para que sólo se muestre el grid "Grid Lab UPM".

```
<llsd>
    <map>
        <key>gridlab upm</key>
        <map>
            <key>LastModified</key>
            <date>2012-10-25T12:33:50.86Z</date>
            <key>grid_login_id</key>
            <undef />
            <key>gridname</key>
            <string>GridLab UPM</string>
            <key>gridnick</key>
            <string>gridlab</string>
            ...
            ...
        </map>
    </map>
</llsd>
```

- Se modificó la clave ***GridListDownload*** en el archivo *NEWVIEW_PATH\app_settings\settings.xml* para evitar que se descargue el listado de Grid del servidor *http://phoenixviewer.com/app/fsdata/grids.xml*.

```
<key>GridListDownload</key>
  <map>
    <key>Comment</key>
    <string>Whether to fetch a grid list from the URL specified in ↩
        GridListDownloadURL.</string>
    <key>Persist</key>
    <integer>1</integer>
    <key>Type</key>
    <string>Boolean</string>
    <key>Value</key>
    <integer>0</integer>
  </map>
```

Temas

Como penúltimo punto, se decidió modificar el *look and feel* de Firestorm; se optó por elegir el tema **OficialV3** ya que tenía una apariencia más agradable, pero, sobre todo, porque el menú contextual de los objetos, se mostraba en forma de listado y no se tenía que presionar un botón "Más" para poder visualizar otras opciones. Elegido el tema por defecto, se realizaron los siguientes cambios:

- Para mantener el tema ***OficialV3*** por defecto, se modificó el valor de la clave ***SessionSettingsFile*** en el archivo *NEWVIEW_PATH\llappviewer.cpp*, tal como sigue:

```
        //Aproximadamente línea número 2647.
        gSavedSettings.setString("SessionSettingsFile", "settings\_v3.xml");
```

- Se ocultaron los demás temas disponibles (Firestorm, Phoenix e Hibrid), comentando las líneas del archivo XML, *NEWVIEW_PATH\skins\default\xui\en\panel_login.xml*, tal como sigue:

```
    <combo_box
          follows="right|bottom"
          top="35"
          left_pad="10"
          right="-15"
          height="23"
          max_chars="128"
          tool_tip="Select which style of viewer you are most familiar with to ↩
              set your defaults to appropriately."
          name="mode_combo"
          width="110">
           <!--<combo_box.item
            label="Firestorm"
            name="Firestorm"
            value="settings_firestorm.xml" />
           <combo_box.item
            label="Phoenix"
            name="Phoenix"
            value="settings_phoenix.xml" />-->
           <combo_box.item
            label="V3"
            name="V3"
            value="settings_v3.xml" />
           <!--<combo_box.item
            label="Hybrid"
            name="Hybrid"
            value="settings_hybrid.xml" />-->
        </combo_box>
```

Luego de que ya se había configurado para que ***OficialV3*** sea el tema por defecto de Firestorm, se modificaron las opciones innecesarias del menú contextual del objeto, dado que el usuario nunca las iba a utilizar en el desarrollo de la práctica. Se modificó el archivo XML *NEWVIEW_PATH\skins\default\xui\en\menu_pie_object.xml* al que se le comentaron los nodos cuyo valor en el atributo ***"label"*** era *"Create"*, *"Sit Here"*, *"Buy"*, *"Take"*, *"Pay"* o *"More"*.

4.2.5. Claridad de los mensajes

Como punto final se decidió cambiar el estilo de las fuentes y transparencias de los mensajes de chat o mensajes que el mismo tutor enviaba. Para conseguir esto se modificó lo siguiente:

- Semodificóel valor de la clave ***ObjectIMColor***, del archivo *NEWVIEW_PATH\skins\default\colors.xml*, para que el color de la fuente de los mensajes del sistema sea de color blanco.

```
      <color
     name="ObjectIMColor"
     reference="White" />
```

- Se modificó el valor de la clave ***ChatFontSize***, del archivo *NEWVIEW_PATH\app_settings\settings.xml*, correspondiente al tamaño de fuente de la ventana de chat.

```
<key>ChatFontSize</key>
    <map>
        <key>Comment</key>
        <string>Size of chat text in chat floater (0 to 3, small to huge)</string↩
            >
        <key>Persist</key>
        <integer>1</integer>
        <key>Type</key>
        <string>S32</string>
        <key>Value</key>
        <integer>2</integer>
    </map>
```

- Se modificó el valor del atributo ***reference***, del archivo *NEWVIEW_PATH\skins \starlight\themes \original_orange\colors.xml*, a ***"White"*** para que el color de la fuente sea blanco.

```
        <color
     name="ObjectIMColor"
     reference="White" />
```

- Se modificó el valor de la clave ***InactiveFloaterTransparency***, del archivo *NEW-VIEW_PATH\app_settings\settings.xml*, para que la ventana sea totalmente opaca.

```
<key>InactiveFloaterTransparency</key>
    <map>
      <key>Comment</key>
      <string>Transparency of inactive floaters (floaters that have no focus)</↩
          string>
      <key>Persist</key>
      <integer>1</integer>
      <key>Type</key>
      <string>F32</string>
      <key>Value</key>
      <real>1</real>
    </map>
```

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

Los entornos virtuales ofrecen muchas posibilidades para el desarrollo de sistemas educativos, una de las principales ventajas es que permiten una mayor inmersión por parte de los alumnos y, de alguna manera, los límites que se tienen dentro de una educación tradicional pueden ser superados. Aprovechar todos los recursos tecnológicos de los que se disponen actualmente, ayuda a mejorar la calidad de la enseñanza y de la participación y discusión entre los alumnos.

Dado que la enseñanza en un entorno real incurre en elevados costos debido a los materiales utilizados o a la disponibilidad de determinadas instalaciones, en una enseñanza por medio de los mundos virtuales, estos costos se reducen considerablemente; por ejemplo, en un laboratorio de química, física, o en general, en un laboratorio en el que se realicen experimentos científicos, el uso de instrumentos y/o sustancias está siempre presente; dado que todos los estudiantes necesitan realizar las prácticas, se necesita una gran cantidad de estos elementos, por lo que los costos ascienden por cada vez que se realicen las prácticas. Sin embargo, en un laboratorio virtual, estos ejercicios pueden ser realizados ilimitadamente sin caer en esos costos y con un aprendizaje mucho más efectivo debido a que pueden practicar varias veces y aprender por ensayo y error.

Uno de los objetivos que se persiguen al construir un laboratorio virtual es conseguir un realismo similar al de un mundo real, es decir, se debe de tratar que el estudiante pueda sentirse lo más cómodo posible al interactuar con los elementos del medio. Los objetos que lo componen deben de guardar cierta similitud en formas, tamaños, colores u otros aspectos, con los reales y, además, se deben de tratar de igualar los gestos que realiza el avatar, como tocar un objeto, soltarlo, lanzarlo, usar ambas manos, etc. La interacción con otros avatares también debe considerarse, por lo tanto, se deben de aprovechar los

recursos de chat de texto, chat de audio y conversaciones multi-usuario que proporcionan las plataformas de los mundos virtuales.

Gracias a las alternativas tecnológicas actuales, es posible disponer de plataformas de desarrollos de sistemas educativos en mundos virtuales de una manera relativamente sencilla. Second Life y OpenSimulator ofrecen un potencial inmenso para la construcción de estos entornos, sin embargo, debido a las necesidades de cada sistema, se debe de evaluar cuál se adecúa mejor. Dado que OpenSim es open-source, no tiene costo y permite un mayor control sobre el servidor del mundo, por lo que es la elección mayoritaria entre las instituciones educativas.

Para el desarrollo de la Transformación Bacteriana en el presente trabajo, se reutilizaron muchos de los componentes ya existentes en el laboratorio virtual de Biotecnología; el mayor provecho que se obtuvo fue el de disponer de un tutor automático que validaba todas las acciones de los estudiantes que se encontraban en su archivo de configuración de la práctica virtual. Gracias a este tutor, en muchos casos, sólo se necesitaba añadir la nueva acción a realizar dentro del *script* de los objetos como una alternativa más a considerar, similar a otras ya codificadas con anterioridad. Es así como se hicieron modificaciones a algunos de los objetos que ya existían, con el fin de que los cambios no tuvieran impacto en fases previas ya desarrolladas. Por otro lado, el modelado 3D de los objetos requeridos para el proceso de electroporación no fue una tarea difícil y se terminaron en poco tiempo.

Una de las dificultades para el desarrollo de esta práctica era que, dadas las características del lenguaje de *scripts*, no se podía hacer una adecuada reutilización de código. Por ejemplo, acciones como tocar o soltar que eran comunes en varios objetos como la placa petri, una botella o un tubo *Eppendorf*, tenían que ser replicadas en todos sus *scripts* y tan solo se podían hacer algunas variantes para cada caso específico. Dada la arquitectura actual y el lenguaje de desarrollo del laboratorio, no es posible hacer una refactorización para este tipo de acciones, aunque tal vez sea posible hacer una adecuación en el tutor automático para poder mitigar esta redundancia de código, lo que actualmente dificulta su mantenimiento.

En cuanto al visor Firestorm, la ventaja de disponer del código fuente fue de gran utilidad. Sin embargo, el proceso de puesta a punto de la solución de *Microsoft Visual Studio 2010* fue una tarea complicada ya que se necesitaban instalar muchos programas adicionales y configurarlos de tal manera que funcionen correctamente. En la página web del autor se explicaba este proceso, pero algunos detalles no se explicaban correctamente o no funcionaban, por tanto, se tuvieron que hacer muchos artificios para poder obtener la solución con el código completo operativo.

Cuando se obtuvieron los archivos fuente, las modificaciones sobre ellos fue una tarea

relativamente sencilla, ya que en la estructura de la solución, los ficheros estaban muy bien separados y ordenados; existían carpetas para los ficheros XML, para los archivos de cabecera y para las clases C++. El único inconveniente fue que los archivos de otros idiomas u otros skins no se encontraban visibles desde el *Microsoft Visual Studio 2010*, por lo que se tenía que ir buscándolos navegando entre las carpetas. Luego de que se hicieron las modificaciones indicadas en el capítulo "Planteamiento del Problema", se realizaron diversas pruebas en varios equipos mediante la realización de las fases de la práctica ya desarrolladas. En el transcurso de estas pruebas se observaron algunos casos no considerados y que fueron corregidos posteriormente.

5.2. Trabajo Futuro

Como trabajos futuros, se sugiere investigar acerca del uso de otros lenguajes de desarrollo como C# o VB .NET para facilitar la reutilización de código fuente. Se podrían crear librerías externas y registrarlas en el .NET Framework y que puedan ser reutilizadas desde el mundo virtual; además, el uso de sus propias librerías puede hacer posible que el método para guardar y leer los archivos planos sea más rápido, dado que los notecards que se tienen en los objetos son archivos de tipo binario que se guardan en la base de datos de OpenSim; como consecuencia, el tiempo de ejecución del algoritmo del tutor puede disminuir sensiblemente cuando se accede a *notecards*. Además, la configuración del tutor se debería de hacer a través de una interfaz externa, ya que los mensajes pueden cambiar con el tiempo, así como los nombres de las sustancias de las vitrinas de instrumentos o cualquier otro texto que se encuentra como *hardcode* dentro del código LSL.

En cuanto a la Transformación Bacteriana, quedan pendientes las pruebas con los alumnos y profesores reales, tal como se hicieron con las fases anteriores.

Parte del desarrollo de este documento se realizó gracias al estudio previo hecho en [Riofrío Luzcano, 2012]. Gracias a los avances hechos en el tutor, en los objetos y en su funcionamiento, al realizar la nueva fase se verificó que las funcionalidades del tutor pueden ser aprovechadas fácilmente para nuevos casos de uso.

Apéndice A

Documentación Adicional

A.1. Procedimiento de descarga del código fuente de Firestorm

Los nombres de algunas carpetas se describirán en la Sección A.2.

A.1.1. Instalación de requisitos

1. Acceder a la página oficial de ***Second Life Knowledge Base*** a través de "http://wiki.secondlife.com/wiki/Viewer_2_Microsoft_Windows_Builds" e instalar todo lo mencionado en la sección *"Establish your programming environment"*.

2. En el caso de software Microsoft se requiere:

   - Microsoft Visual C++ 2010 Redistributable Package (En caso de error, verificar que no se tenga instalada una versión de 32 ó 64 bits. En caso se encuentre, se deber desinstalar y reiniciar el equipo para continuar).
   - Windows SDK for Windows 7 and .NET Framework 4.
   - DirectX SDK (Junio, 2010).
   - Microsoft Visual Studio 2010 Service Pack 1.

3. Para el caso de software adicional se requiere:

   - CMake (http://www.cmake.org/);
   - Python 2.7 (http://www.python.org/download/).
   - Mercurial TortoiseHg (http://tortoisehg.bitbucket.org/).

- Cygwin, verificar que tambin se instalen los utilitarios Flex y BISON (http://cygwin.com/install.html).
- Unicode NSIS (http://www.scratchpaper.com/).

A.1.2. Instalación de *Autobuild*

- Abrir el símbolo de sistema de Windows y navegar hasta la carpeta FSPATH.
- Ejecutar el siguiente comando :
  hg clone http://hg.secondlife.com/autobuild/
- En las Variables de Entorno de Windows, crear la variable AUTOBUILD_VSVER y asignarle el valor de 100.

A.1.3. Configuración de *Microsoft Visual Studio 2010*

Acceder a la página oficial de ***Second Life Knowledge Base*** a través de "http://wiki.secondlife.com/wiki/Viewer_2_Microsoft_Windows_Builds" e instalar todo lo mencionado en la sección *"Configure VC2010"*.

A.1.4. Descarga de estructura de Firestorm

1. Abrir el símbolo de sistema de Windows y navegar hasta la carpeta FSPATH.
2. Ejecutar el siguiente comando :
   hg clone http://hg.phoenixviewer.com/phoenix-firestorm-lgpl/

A.1.5. Pasos adicionales

1. Abrir el símbolo de sistema de Windows y navegar hasta la carpeta FSPATH.
2. Ejecutar el siguiente comando :
   hg clone https://bitbucket.org/lindenlab/3pfmod/ y luego,
   cd 3pfmod
3. En este punto, ubicado en la carpeta *3pfmod*, copiar el archivo *fmodapi375win.zip* y descomprimir. A la carpeta descomprimida cambiarle de nombre a *fmodapi375*.
4. Navegar hasta el archivo : *FSPATH*
   phoenix-firestorm-lgpl

3p-fmod
build-cmd.sh

5. Comentar la línea :
   fetch_archive "$FMOD_URL" "$FMOD_ARCHIVE" "$FMOD_MD5".

6. Comentar la línea :
   unzip -n "$FMOD_ARCHIVE" << "$FMOD_ARCHIVE".unzip-log 2¿&1

7. Volver al símbolo del sistema y ejecutar el siguiente comando:
   autobuild build all

8. Ejecutar el siguiente comando:
   autobuild package

9. Descomentar las líneas que anteriormente fueron comentadas.

A.1.6. Descarga de fuentes

1. Abrir el símbolo de sistema de Windows y navegar hasta la carpeta FSPATH.

2. Ejecutar el siguiente comando :
   cd phoenix-firestorm-lgpl

3. Ejecutar el siguiente comando :
   autobuild configure -c DebugOS

4. Al ejecutar este comando todos los archivos fuente deberían de descargarse en la carpeta *phoenix-firestorm-lgpl/build-vc100*

5. Abrir el archivo Firestorm.sln

A.2. Listado de carpetas de Firestorm modificadas

Carpeta	**Descripción**
FSPATH	Carpeta inicial en la que se descargan todos los fuentes de Firestorm.
NEWVIEW_PATH	Carpeta raiz en la que se encuentran los ficheros fuente. **Ubicación** : FSPATH\indra\newview\

XML_PATH	Carpeta en la que se encuentran los ficheros XML de configuración. **Ubicación** : NEWVIEW_PATH\app_settings\
COLORS_PATH	Carpeta en la que se encuentran los ficheros XML de los temas/skins. **Ubicación** : NEWVIEW_PATH\skins\default\
USER_PATH	Carpeta en la que se encuentran los ficheros XML con las configuraciones individuales de los usuarios. **Ubicación** : *%APPDATA %*\Firestorm*[Usuario_Firestorm]*\, en donde, *%APPDATA %* representa la carpeta "Datos de programa" del usuario de Microsoft Windows actual y [Usuario_Firestorm], el usuario autenticado en el laboratorio de Biotecnología.
XUI_ES_PATH	Carpeta en la que se encuentran los ficheros XML del tema **Oficial V3**, versión en Español. **Ubicación** : NEWVIEW_PATH\skins\default\xui\es\

A.3. Estructura de *LLWearObjectPanel*

Elemento	**Descripción**
mWearButton	Propiedad de tipo **LLButton** que representa al botón "Soltar" .
gUUID	Propiedad de tipo ***LLUUID*** que almacena el identificador del objeto adjuntado.
ShowUnwearButton(bool)	Método que muestra u oculta el botón "Soltar" .
ShowUnwearButton(bool, LLUUID)	Método que muestra el botón "Soltar" al momento de adjuntar un objeto a la mano de avatar.
getInstance()	Método que devuelve la única instancia estática de ***LLWearObjectPanel***.
getAttachedObject()	Método que devuelve el valor de gUUID.
postBuild()	Método en el que se construye y agrega el botón "Soltar" al panel ***LLWearObjectPanel***.
setVisible()	Método que muestra u oculta el panel ***LLWearObjectPanel***.

handleToolTip(S32,S32,MASK)	Método manejador de tooltips.
reparent(LLFLoaterMove)	Método que agrega el panel ***LLWearObjectPanel*** al contenedor de paneles.
LLWearObjectPanel()	Constructor de la clase ***LLWearObjectPanel***.
getWearObjectPanel()	Método que construye el panel ***LLWearObjectPanel***.
onUnWearButtonClick()	Método que se ejecuta cuando se presiona el botón "Soltar" .
updatePosition()	Método que actualiza la posición del panel en el visor.

Apéndice B

Codificaciones

B.1. Clase LLWearObjectPanel

```
LLWearObjectPanel::LLWearObjectPanel() :
    mWearButton(NULL)
{
    // make sure we have the only instance of this class
    static bool b = true;
    llassert_always(b);
    b=false;
}

void LLWearObjectPanel::ShowUnwearObjectButton(BOOL visible)
{
    LLWearObjectPanel* panel = getInstance();
    panel->mWearButton->setVisible(visible);
}

void LLWearObjectPanel::ShowUnwearObjectButton(BOOL visible,LLUUID mmUID)
{
    ShowUnwearObjectButton(visible);
    LLWearObjectPanel::gUUID = mmUID;
}

LLWearObjectPanel* LLWearObjectPanel::getInstance()
{
    static LLWearObjectPanel* panel = getWearObjectPanel();
    return panel;
}

LLUUID LLWearObjectPanel::getAttachedObject()
{
    return LLWearObjectPanel::getInstance()->gUUID;;
}

void LLWearObjectPanel::setAttachedObject(LLUUID uid)
{
```

```
    gUUID = uid;
}

LLWearObjectPanel* LLWearObjectPanel::getWearObjectPanel()
{
    LLWearObjectPanel* panel = new LLWearObjectPanel();
    panel->buildFromFile("panel_wear_object.xml");

    llinfos << "Build LLWearObjectPanel panel" << llendl;

    panel->updatePosition();
    return panel;
}

void LLWearObjectPanel::updatePosition()
{
    return;
}

BOOL LLWearObjectPanel::postBuild()
{
    mWearButton = getChild<LLButton>("unwear_btn");
    mWearButton->setCommitCallback(boost::bind(&LLWearObjectPanel::onUnWearButtonClick,↩
        this));
    mWearButton->setVisible(FALSE);
    return TRUE;
}


void callback_attachment_drop_custom(const LLSD& notification, const LLSD& response)
{
    S32 option = LLNotificationsUtil::getSelectedOption(notification, response);
    if (option != 0) return;

    LLUUID object_id;
    object_id = LLWearObjectPanel::getAttachedObject();
    LLInventoryItem* item = gInventory.getItem(object_id);

    LLViewerObject *object = gAgentAvatarp->getWornAttachment(item->getUUID());

    if (!object)
    {
        llwarns << "handle_drop_attachment() - no object to drop" << llendl;
        return;
    }

    LLViewerObject *parent = (LLViewerObject*)object->getParent();
    while (parent)
    {
        if(parent->isAvatar())
        {
            break;
        }
        object = parent;
        parent = (LLViewerObject*)parent->getParent();
```

```
    }

    if (!object)
    {
        llwarns << "handle_detach() - no object to detach" << llendl;
        return;
    }

    if (object->isAvatar())
    {
        llwarns << "Trying to detach avatar from avatar." << llendl;
        return;
    }

    // reselect the object
    LLSelectMgr::getInstance()->selectObjectAndFamily(object);
    LLSelectMgr::getInstance()->sendDropAttachment();

    LLWearObjectPanel::getInstance()->mWearButton->setVisible(FALSE);
    return;
}

void LLWearObjectPanel::onUnWearButtonClick()
{
    LLSD payload;
    LLNotificationsUtil::add("AttachmentDrop", LLSD(), payload, &↩
        callback_attachment_drop_custom);
}

void LLWearObjectPanel::setVisible(BOOL visible)
{
    //we dont need to show the panel if these buttons are not activated
    if (gAgentCamera.getCameraMode() == CAMERA_MODE_MOUSELOOK) visible = false;

    if (visible)
    {
        updatePosition();
    }


    if (getParent()) getParent()->setVisible(visible);


    // also change own visibility to avoid displaying the panel in mouselook
    //(broken when EXT-2504 was implemented).
    LLPanel::setVisible(visible);
}


BOOL LLWearObjectPanel::handleToolTip(S32 x, S32 y, MASK mask)
{
    LLToolTipMgr::instance().unblockToolTips();

    if (mWearButton->getVisible())
    {
```

```
        LLToolTipMgr::instance().show(mWearButton->getToolTip());
    }

    return LLPanel::handleToolTip(x, y, mask);
}

void LLWearObjectPanel::reparent(LLFloaterMove* move_view)
{
    LLPanel* parent = dynamic_cast<LLPanel*>(getParent());
    if (!parent)
    {
        return;
    }

    if (move_view != NULL)
    {
        llassert(move_view != parent); // sanity check

        // Save our original container.
        if (!mOriginalParent.get())
            mOriginalParent = parent->getHandle();

        // Attach to movement controls.
        parent->removeChild(this);
        move_view->addChild(this);
        // Origin must be set by movement controls.

    }
    else
    {
        if (!mOriginalParent.get())
        {
            llwarns << "Original parent of the stand / stop flying panel not found" << ↩
                llendl;
            return;
        }

        // Detach from movement controls.
        parent->removeChild(this);
        mOriginalParent.get()->addChild(this);
        // update parent with self visibility (it is changed in setVisible()). EXT-4743
        mOriginalParent.get()->setVisible(getVisible());


        updatePosition(); // don't defer until next draw() to avoid flicker
    }

    // <FS:Zi>Make sure to resize the panel to fit the new parent. Important for
    // proper layouting of the buttons. Skins should adapt the parent container.
    if(getParent())
        reshape(getParent()->getRect().getWidth(),getParent()->getRect().getHeight(),↩
            FALSE);
    // </FS:Zi>
}
```

B.2. Visibilidad del botón "Soltar" desde clases externas

Las clases que se mencionan a continuación contienen métodos que fueron adecuados para mostrar/ocultar el botón "Soltar" de acuerdo a las validaciones del punto "Validaciones adicionales".

B.2.1. Clase LLAgentWearables

```
void LLAgentWearables::userUpdateAttachments(LLInventoryModel::item_array_t& ↩
    obj_item_array, bool fAttachOnly)
// [/SL:KB]
{
    ...
    if (requested_item_ids.find(object_item_id) != requested_item_ids.end())
    {
        // Object currently worn, was requested.
        // Flag as currently worn so we won't have to add it again.
        current_item_ids.insert(object_item_id);

        //=============================================================================
        //Se agregó la siguiente línea para que el botn ''Soltar'' se muestre siempre y
        //cuando el avatar tenga adjuntado un objeto al momento de inicializar el visor.
        LLWearObjectPanel::getInstance()->ShowUnwearObjectButton(TRUE,object_item_id);
        //=============================================================================
    }
    ...
}
```

B.2.2. Clase LLAppearanceMgr

```
void LLAppearanceMgr::onRegisterAttachmentComplete(const LLUUID& idItem)
{
    const LLUUID& idItemBase = gInventory.getLinkedItemID(idItem);

    // Remove the attachment from the pending list
    uuid_vec_t::iterator itPendingAttachLink = std::find(mPendingAttachLinks.begin(), ↩
        mPendingAttachLinks.end(), idItemBase);
    if (itPendingAttachLink != mPendingAttachLinks.end())
        {
            //=============================================================================
            //Se agregó la siguiente línea para que el botn ''Soltar'' se muestre luego de que el
            //objeto se adjunte nuevamente al avatar cuando exista una validacin en el tutor.
            LLWearObjectPanel::getInstance()->ShowUnwearObjectButton(TRUE,idItemBase);
            //=============================================================================

            mPendingAttachLinks.erase(itPendingAttachLink);
```

```
        }

    // It may have been detached already in which case we should remove the COF link
    if ( (isAgentAvatarValid()) && (!gAgentAvatarp->isWearingAttachment(idItemBase)) )
        removeCOFItemLinks(idItemBase, false);

}
```

B.2.3. Clase LLAttachmentsMgr

```
void LLAttachmentsMgr::addAttachment(const LLUUID& item_id,
                                     const U8 attachment_pt,
// const BOOL add)
// [RLVa:KB] - Checked: 2010-09-13 (RLVa-1.2.1c) | Added: RLVa-1.2.1c
                                     const BOOL add, const BOOL fRlvForce /*=FALSE*/)
// [/RLVa:KB]
{
    AttachmentsInfo attachment;
    attachment.mItemID = item_id;
    attachment.mAttachmentPt = attachment_pt;
    attachment.mAdd = add;

    //============================================================================
    //En caso el objeto sea atachado, se muestra el botón ‘‘Soltar’’.
    LLWearObjectPanel::getInstance()->ShowUnwearObjectButton(TRUE,item_id);
    //============================================================================
    ...
}
```

B.2.4. Clase LLObjectBridge

```
void LLObjectBridge::performAction(LLInventoryModel* model, std::string action)
{
    //============================================================================
    LLUUID object_id_= mUUID;
    //Se agregó la siguiente lńea para ocultar el botón ‘‘Soltar’’ cuando
    //se atache un nuevo objeto al avatar y para que se oculte en caso
    //se produzcan acciones distintas a ‘‘attach’’.
    LLWearObjectPanel::getInstance()->ShowUnwearObjectButton(FALSE);
    //============================================================================
    ...
}
```

B.2.5. Clase LLAttachmentDetach

```
bool handleEvent(const LLSD& userdata)
{
    ...
    if (LLSelectMgr::getInstance()->getSelection()->isAttachment())
    {
        LLSelectMgr::getInstance()->sendDetach();
        //========================================================================
        //Se agregó la siguiente línea para que el botón ‘‘Soltar’’ se oculte cuando se pulse la
        //opción ‘‘Quitar’’ del menú que aparece cuando se hace click derecho sobre el objeto.
        LLWearObjectPanel::getInstance()->ShowUnwearObjectButton(FALSE);
        //========================================================================
    }
    return true;
}
```

Bibliografía

[Allison et al., 2010] Allison, C., Miller, A., Sturgeon, T., Nicoll, J. R., and Perera, I. (2010). Educationally enhanced virtual worlds.

[Andreas et al., 2010] Andreas, K., Thrasyvoulos, T., Stavros, D., and Andreas, P. (2010). Collaborative learning in opensim by utilizing sloodle.

[Bartle, 2003] Bartle, R. A. (2003). *Designing Virtual Worlds*. New Riders.

[Berns et al., 2013] Berns, A., Gonzalez-Pardo, A., and Camacho, D. (2013). Game-like language learning in 3-d virtual environments. *Computers and Education*, 60(1):210 – 220.

[Castronova, 2011] Castronova, E. (2011). Synthetic worlds: the business and culture of online games.

[Chawla, 2002] Chawla, H. S. (2002). *Introduction to Plant Biotechnology*. Science Publishers, Incorporated.

[Clark and Pazdernik, 2009] Clark, D. and Pazdernik, N. (2009). *Biotechnology: Applying the Genetic Revolution*. Academic Press. Academic Press/Elsevier.

[Datli et al., 2009] Datli, E. L., Vegoudakis, K. I., Pappas, C., and Bamidis, P. D. (2009). Re-use and exchange of an opensim platform based learning environment among different medical specialties for clinical scenarios. In *Information Technology and Applications in Biomedicine, 2009. ITAB 2009. 9th International Conference on*.

[Imprudence, 2013] Imprudence (2013). Kokua/imprudence blog - news about the kokua and imprudence viewers. http://blog.kokuaviewer.org/.

[Inc., 2013] Inc., B. E. (2013). World of warcraft. http://eu.battle.net/wow/es/.

[Inc, 2013a] Inc, E. A. (2013a). The sims. http://thesims.com/.

[Inc, 2013b] Inc, L. R. (2013b). Second life. http://secondlife.com/.

[Inc., 2013] Inc., T. P. F. P. (2013). Firestorm viewer - the phoenix firestorm project inc. http://www.firestormviewer.org/.

[Life, 2013] Life, S. (2013). Lsl portal. http://wiki.secondlife.com/wiki/LSL_Portal.

[Lorenzo et al., 2012] Lorenzo, C., Lezcano, L., Vecino, C., and Sicilia, M. A. (2012). Slroute: Learning spanish in immersive environments through the way of st. james. In *Computers in Education (SIIE), 2012 International Symposium on*.

[Ltd, 2013] Ltd, J. (2013). Rune scape. http://www.runescape.com/.

[María, 2013] María, K. (2013). U.s. army goes virtual with opensim.

[OpenSim, 2013a] OpenSim (2013a). History opensim. http://opensimulator.org/wiki/History.

[OpenSim, 2013b] OpenSim (2013b). Opensim home. http://opensimulator.org/.

[OpenSimulator, 2013a] OpenSimulator (2013a). Configuration. http://opensimulator.org/wiki/Configurations.

[OpenSimulator, 2013b] OpenSimulator (2013b). Scripting languages. http://opensimulator.org/wiki/Scripting_Languages.

[Pedraza, 2013] Pedraza, D. F.-A. (2013). Diseño de una práctica para un laboratorio virtual de biotecnología multilingüe y adaptable a alumnos de secundaria. Trabajo de fin de grado, Facultad de Informática - Universidad Politécnica de Madrid, Madrid - España.

[Radegast, 2013] Radegast (2013). Radegast metaverse client. http://radegast.org/wiki/Radegast.

[Riofrío Luzcano, 2012] Riofrío Luzcano, D. (2012). Diseño e implementación de un laboratorio virtual de biotecnología. Trabajo de fin de máster, Facultad de Informática - Universidad Politécnica de Madrid, Madrid - España.

[Rymaszewski, 2007] Rymaszewski, M. (2007). *Second Life: The Official Guide*. Wiley.

[Savin-Baden, 2010] Savin-Baden, M. (2010). *A Practical Guide to Using Second Life in Higher Education*. McGraw-Hill Education.

[Singularity, 2013] Singularity (2013). Singularity viewer. http://www.singularityviewer.org/.

[Studio, 2013] Studio, G. I. I. (2013). Gaia. http://www.gaiaonline.com/.

[Sun et al., 2010] Sun, B., Wu, H., Zhao, H., and Hu, X. (2010). Research and application on plug-in technology in opensim. In *Audio Language and Image Processing (ICALIP), 2010 International Conference on*.

[Wikimedia, 2013] Wikimedia, F. (2013). Wikipedia - proteobacterias. http://es.wikipedia.org/wiki/Proteobacteria.

[Wikipedia, 2013a] Wikipedia (2013a). Electroporación. http://es.wikipedia.org/wiki/Electroporaci

[Wikipedia, 2013b] Wikipedia (2013b). Tubo de microcentrífuga. http://es.wikipedia.org/wiki/Tubo_de_microcentr

[Wood et al., 2001] Wood, D. W., Setubal, C., J., Rajinder, K., Monks, and Dave E., e. a. (2001). The genome of the natural genetic engineer agrobacterium tumefaciens c58. *American Association for the Advancement of Science*, pages 2317–2323.